Eclipse Scripting API

Reference Guide

Eclipse

P1026484-002-B SEPTEMBER 2019

Legal Information

Publication ID

P1026484-002-B

Publication Title

Eclipse Scripting API Reference Guide

Abstract

This document provides reference information and procedures for Eclipse Scripting API.

This publication is the English-language original.

Trademarks

ARIA® Oncology Information System for Radiation Oncology, Varian®, and VMS® are registered trademarks, and Eclipse™ and BrachyVision™ are trademarks of Varian Medical Systems, Inc. Microsoft®, Windows®, .NET ®, Visual Studio®, Visual C#®, and IntelliSense® are registered trademarks of Microsoft Corporation in the United States and other countries.

All other trademarks or registered trademarks are the property of their respective owners.

Notice

Information in this publication is subject to change without notice and does not represent a commitment on the part of Varian. Varian is not liable for errors contained in this publication or for incidental or consequential damages in connection with furnishing or use of this material.

Electronic labeling

This symbol on the label indicates that the Instructions for Use for the corresponding product are available at www.MyVarian.com. Access the Instructions for Use in electronic form by logging in with your assigned MyVarian user credentials.

In compliance with Anvisa and EU regulations, Varian will send Brazil and EU customers a free printed copy of the Instructions for Use within 7 days. Use the "Paper Publication Request" form provided on the Varian webpage to order your copy.

Legal Manufacturer

Varian Medical Systems, Inc.
3100 Hansen Way
Palo Alto, CA 94304-1038
United States of America

FDA 21 CFR 820 Quality System Regulations (cGMPs)

Varian Medical Systems, Oncology Systems products are designed and manufactured in accordance with the requirements specified within this federal regulation.

CAUTION: US Federal law restricts this device to sale by or on the order of a physician.

WHO

ICD-O codes and terms used by permission of WHO, from:

- International Statistical Classification of Diseases and Related Health Problems, Tenth Revision (ICD-10) 2008 Edition. Vols 1-3. Geneva, World Health Organization, 2008.

ICD-10 codes and terms used by permission of WHO, from:

- International Classification of Diseases for Oncology, Third Edition.

EU REACH SVHC Disclosure

The link to the current EU REACH SVHC disclosure statement can be found at http://www.varian.com/us/corporate/legal/reach.html

International Organization for Standardization ISO 13485

Varian Medical Systems Oncology Systems products are designed and manufactured in accordance with the requirements specified within the ISO 13485 quality standard.

Medical Device

IEC 62083

Eclipse™ Treatment Planning System is IEC 62083 compliant.
BrachyVision™ Treatment Planning System is IEC 62083 compliant.

Authorized Representative in the EU

Varian Medical Systems Netherland B.V.
Kokermolen 2
3994 DH Houten
The Netherlands

CE 2797

Table of Contents

Introduction

Eclipse is used to plan radiotherapy treatments for patients with malignant or benign diseases. The users of Eclipse are medical professionals who have been trained in radiation dosimetry. After an oncologist has decided that radiotherapy is the suitable treatment for a patient, the medical professionals use Eclipse to plan the treatment for the patient. Eclipse can be used to plan external beam irradiation with photon, electron, and proton beams, as well as for internal irradiation (brachytherapy) treatments. Eclipse is part of Varian's integrated oncology environment.

The Eclipse Scripting Application Programming Interface (Eclipse Scripting API or ESAPI) is a programming interface and a software library for Eclipse. It allows software developers to write scripts to access the treatment planning information in Eclipse. The scripts can be integrated into the Eclipse user interface, or they can be run as stand-alone executables.

Who Should Read This Manual

This manual is written mainly for medical/technical personnel who wish to write custom scripts to be used in Eclipse. It is assumed that you are familiar with:

- Eclipse Treatment Planning System
- Radiation oncology domain and concepts
- DICOM
- Software engineering practices
- Microsoft Visual Studio development environment
- Microsoft Visual C# programming language and object oriented development

Visual Cues

This publication uses the following visual cues to help you find information:

WARNING:	A warning describes actions or conditions that can result in serious injury or death.
CAUTION:	A caution describes hazardous actions or conditions that can result in minor or moderate injury.
NOTICE:	A notice describes actions or conditions that can result in damage to equipment or loss of data.

Note: A note describes information that may pertain to only some conditions, readers, or sites.

Tip: A tip describes useful but optional information such as a shortcut, reminder, or suggestion, to help get optimal performance from the equipment or software.

Related Publications

RT Administration Reference Guide	P1026480-001-A
Beam Configuration Reference Guide	P1026479-001-A
BrachyVision Instructions for Use	P1026473-001-A
BrachyVision Reference Guide	P1026474-001-A
BrachyVision Algorithms Reference Guide	P1026482-001-A
Eclipse Photon and Electron Algorithms Reference Guide	P1026471-001-A
Eclipse Cone Planning Online Help	P1033300-001-A
Eclipse Photon and Electron Instructions for Use	P1026458-001-A
Eclipse Photon and Electron Reference Guide	P1026470-001-A
Eclipse Proton Reference Guide	P1026478-001-A
Eclipse Proton Instructions for Use	P1026477-001-A
Eclipse Proton Algorithms Reference Guide	P1026475-001-A
Eclipse Scripting API Online Help	P1026485-001-A

Contact Varian Customer Support

For user documentation or customer support, go to www.MyVarian.com.

About The Eclipse Scripting API

The Eclipse Scripting API is a Microsoft .NET class library that gives you access to the treatment planning data of Eclipse. It allows you to create scripts that leverage the functionality of Eclipse, and lets you retrieve plan, image, dose, structure, and DVH information from the Varian System database. The data is retrieved from the Varian System database also in stand-alone Eclipse installations. You can integrate the scripts into Eclipse, or you can run them as stand-alone executables.

With Eclipse Automation feature, you can also create scripts that allow you to create and modify structure and plan data, and execute dose calculation and optimization algorithms. These scripts are first created and tested in a non-clinical development environment, but can then be approved for clinical use.

WARNING: The authors of custom scripts are responsible for verifying the accuracy and correctness of the scripts after developing a new script or after system upgrade for the existing scripts.

Features

By using the Eclipse Scripting API, you can:

- Write custom scripts and integrate them into the Eclipse user interface.
- Write stand-alone executable applications that leverage the Eclipse Scripting API.

You can access the following information with ESAPI scripts:

- Image and structure models, including their volumetric representations.
- Plans, fields, and accessories.
- Predecessor plans.
- Plan protocol information.
- IMRT optimization objectives and parameters.
- Clinical goals
- Doses, including their volumetric representations.
- Dose volume histograms.
- Optimal fluences.
- DVH estimates.
- Plan uncertainty information.
- Prescription Information.
- Treatment session information.

With Eclipse Automation, you can also create scripts that:

- Create and modify structures and structure sets.
- Create and modify plans and fields.

- Create and modify verification plans and copy images from another patient for those plans (for example, copy a scanned phantom image from a designated case).
- Create artificial phantom images.
- Generate DRRs.
- Create evaluation doses to evaluate dose calculated outside of Eclipse.
- Optimize plans by using the Eclipse optimization algorithms.
- Calculate leaf motions by using the Eclipse leaf motion calculation algorithms.
- Calculate dose by using the Eclipse dose calculation algorithms.
- Execute DVH estimation.
- Modify raw and final scan spot lists for proton plans.

The Eclipse Scripting API provides you also the following:

- Possibility to use visual scripting.
- A wizard that makes it simple to create new scripts.
- Patient data protection that complies with HIPAA.
- Support for user authorization used in Eclipse and ARIA Radiation Therapy Management (RTM).
- API documentation.
- Example applications.
- Full 64-bit support.

System Requirements

The basic system requirements of the Eclipse Scripting API are the same as those of Eclipse. For more information, refer to *Eclipse Customer Release Note*.

Note: Microsoft Visual Studio is not needed for creating scripts. However, some features described in this document assume that Microsoft Visual Studio 2013 has been installed.

Clinical Environment

To run read-only ESAPI scripts in a clinical environment, you need the following:

- Eclipse 15.1 or later.
- A license for the Eclipse Scripting API 15.1 or later.

To approve and run ESAPI scripts created for Eclipse Automation, you need:

- Eclipse 15.1.1 or later.
- A license for the Eclipse Scripting API 15.1.1 or later.
- A license for Eclipse Automation 15.1.1 or later.

Development / Research Environment

To develop ESAPI scripts in a non-clinical development (research) environment, you need the following:

- Eclipse 15.1 or later (optional for creating scripts, mandatory for running them).
- A non-clinical Varian System database configured for research use.
- Eclipse Scripting API license.
- Eclipse Scripting API for Research Users license.

Version Compatibility

Note: If you use an obsoleted type, property, field, or method, the compiler shows a warning. In this case, the compilation of a single-file plug-in fails. If the script is a binary plug-in or a standalone executable, the compiler shows an error. This happens only if the "Treat warnings as errors" project setting is turned on in Microsoft Visual Studio.

ESAPI 16.0

The Eclipse Scripting API 16.0 is compatible with Eclipse 16.0.

Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

The target framework for the scripts must be Microsoft .NET Framework 4.6.1.

ESAPI 15.5

The Eclipse Scripting API 15.5 is compatible with Eclipse 15.5.

Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

The target framework for the scripts must be Microsoft .NET Framework 4.5.

ESAPI 15.1.1

The Eclipse Scripting API 15.1.1 is compatible with Eclipse 15.1.1.

Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

ESAPI 15.1

The Eclipse Scripting API 15.1 is compatible with Eclipse 15.1.

Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

ESAPI 15.0

The Eclipse Scripting API 15.0 is compatible with Eclipse 15.0.

Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

Incompatibilities between ESAPI 15.0 and ESAPI 13.7:

- In prior versions, for stand-alone executables, the method `Application.CreateApplication` took two parameters, `userid` and `password` to identify a user. In Eclipse Treatment Planning System 15.0, a new security framework is introduced, and the logged-in user is automatically identified.
- Fractionation class has been removed, and its properties and methods have been moved to `PlanSetup` class.
- `OptimizationSetup.AddStructurePointCloudParameter()` method and `Beam.ExternalBeam` property that were previously marked as obsolete are now removed (the latter one is replaced by `Beam.TreatmentUnit`).

ESAPI 13.7

The Eclipse Scripting API 13.7 is compatible with Eclipse 13.7.

Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

Incompatibilities between ESAPI 13.7 and ESAPI 13.6:

- In prior versions, proton plans were represented as `ExternalPlanSetup` types. In version 13.7, a new class hierarchy for proton plans (`IonPlanSetup`) has been added. The consequence of this is that methods with return type `ExternalPlanSetup` that used to return proton plans now do not. The following methods and properties have been changed to return only photon external beam plans:
  - `ScriptContext.ExternalPlansInScope`
  - `ScriptContext.ExternalPlanSetup`
  - `Course.ExternalPlanSetups`

The corresponding methods of `ScriptContext` and `Course` returning `PlanSetup` types continue to return external beam plans, brachytherapy plans, and proton plans as previously.

ESAPI 13.6

The Eclipse Scripting API 13.6 is compatible with Eclipse 13.6.

Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

ESAPI 13.5

The Eclipse Scripting API 13.5 is compatible with Eclipse 13.5.

Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

ESAPI 13.0

The Eclipse Scripting API 13.0 is compatible with Eclipse 13.0.

Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

Incompatibilities between ESAPI ESAPI 13.0 and ESAPI 11.0:

- The type `VMS.TPS.Common.Model.Types.VRect` has been changed to immutable. Scripts that use the set accessors of VRect properties are incompatible with the Eclipse Scripting API 13.0.
- The type `VMS.TPS.Common.Model.ExternalBeam` has been marked as obsolete. It is replaced by the `VMS.TPS.Common.Model.ExternalBeamTreatmentUnit` type.
- The property `VMS.TPS.Common.Model.Beam.ExternalBeam` has been marked as obsolete. It is replaced by the `VMS.TPS.Common.Model.Beam.TreatmentUnit` property.

ESAPI 11.0

The Eclipse Scripting API 11.0 is compatible with Eclipse 11.0.

Assembly Version Numbers

The Eclipse Scripting API is a Microsoft .NET class library that is also called an assembly. The .NET assembly can have several different version numbers. The three version numbers that are used in ESAPI are:

- AssemblyVersion: this is used by Visual Studio and .NET framework during building and at runtime to locate, link, and load the assemblies. The assembly version is visible, for example, in Visual Studio, in the properties of the assembly reference.
- AssemblyFileVersion: this is the version number of the file. It is displayed in Windows Explorer, in the file properties dialog of the assembly.
- AssemblyInformationalVersion: the product version of the assembly. It is displayed in Windows Explorer, in the file properties dialog of the assembly. Product version number is also visible in *Eclipse Scripting API Online Help*.

Starting from ESAPI release 15.0, assemblies have used the .NET strong naming. The strong (or full) name of the assembly consists of name, version, culture, and public key token. Originally, these three version numbers have been the same in ESAPI, but starting from the ESAPI release 15.1, the AssemblyVersion is different. This enables changing the AssemblyVersion based on changes in ESAPI, in comparison to the file and product versions that are automatically updated based on Eclipse release.

The following table contains the version numbers per release starting from ESAPI 15.1 release:

ESAPI Release	AssemblyVersion	AssemblyFileVersion	AssemblyInformationVersion
16.0	1.0.400	16.0	16.0
15.5 MR1	1.0.200	15.5	15.5
15.5	1.0.100	15.5	15.5
5.5	1.0.7	15.1	15.1
15.1	5.	15.1	15.1

Upgrade to ESAPI 16.0

Binary plug-in scripts and stand-alone executable scripts that have been compiled using versions 15.5 and later of Eclipse Scripting API will work after upgrading to the Eclipse Scripting API 16.0 without recompilation since the major version of the API has not been changed, there are no breaking changes, just additions to the API.

Stand-alone scripts that have been compiled using versions older than 15.5 of Eclipse Scripting API do not work after upgrading to the Eclipse Scripting API 16.0. Additionally, binary plug-ins for older versions do not compile after the upgrade.

To make the scripts work with ESAPI 16.0, you need to update the Visual Studio projects to reference the new ESAPI 16.0 assemblies.

Do the following:

1. Open the Eclipse Script Visual Studio project.
2. Expand the References item in the Solution Explorer. You should see the existing references to `VMS.TPS.Common.Model.API` and `VMS.TPS.Common.Model.Types`.
3. Remove both references from the project.
4. Add new references to the ESAPI 16.0 assemblies.

   In the **Add Reference** dialog box, select the **Browse** tab. The assemblies are located under the installation directory of the Eclipse Scripting API, in the API subdirectory.

5. Add references to both `VMS.TPS.Common.Model.API.dll` and `VMS.TPS.Common.Model.Types.dll`.
6. In the properties of the solution, make sure that the Target Framework is NET 4.5.
7. Recompile the project.

What Is New in Eclipse Scripting API 16.0

Many new properties, functions, and classes have been added or changed in ESAPI 16.0. See the detailed documentation in *Eclipse Scripting API Online Help*. The most significant changes and additions are listed below.

Halcyon Support

Eclipse automation for Halcyon machines did not work in earlier versions due to unique planning requirements for Halcyon machines. To enable Eclipse automation for Halcyon, the following features were added to Eclipse Scripting API:

- Add imaging sequence field with `ExternalPlanSetup.AddImagingSetup`.
- Add Halcyon fixed sequence (flattened) fields with new method `ExternalPlanSetup.AddFixedSequenceBeam`.
- Add or remove fixed sequence fields with methods `Beam.AddFlatteningSequence` and `Beam.RemoveFlatteningSequence`.
- Add support (couch) structures with new method `PlanSetup.GenerateCouchStructures`.
- Calculate and optimize plans for Halcyon machines.
- New methods `PlanSetup.IsValidForPlanApproval` can be used for checking that the Halcyon plan has been created properly.

Eclipse Scripting API now also provides data model access to the dual layer Halcyon MLC.

Plan Creation Changes

New method `Course.AddExternalPlanSetup` allow for creation of plans with specified target structure and named reference point.

It is now possible to add setup fields with `ExternalPlanSetup.AddSetupFieldForBeam`.

Improved PlanSum support

It is possible to get the active `PlanSum` with new property `ScriptContext.PlanSum`.

It is now possible to create `PlanSums`, remove `PlanSums`, and edit `PlanSums` with Eclipse Scripting API.

- New method `Course.CreatePlanSum` to create `PlanSum`.

- New method `Course.RemovePlanSum` to remove `PlanSum`.
- New methods `PlanSum.AddItem`, `RemoveItem`, `SetPlanWeight` `SetPlanSumOperation` to edit `PlanSum`.
- Property `PlanSum.Name` has now a setter.

Base Dose Planning

It is now possible to add a base dose to a plan for optimization with the new getter / setter property `PlanSetup.BaseDosePlanningItem`.

Clinical Goals

It is now possible to retrieve clinical goals from plans, plan sums or planning item objects with the `GetClinicalGoals` method. The method returns a list of clinical goal objects that can then be looped over to extract the information for reporting.

Each clinical goal object contains the structure ID, priority, objective, acceptable variation, actual value, and a Boolean value telling if the goal is met or not.

Proton Support

The following proton features were added to Eclipse Scripting API:

A proton plan can be added to a course. You can now use:

- `Course.AddIonPlanSetup` for creating a proton treatment plan.
- `Course.AddIonPlanSetupAsVerificationPlan` for creating a proton verification plan.

Modulated scanning fields can now be added to a proton plan using `IonPlanSetup.AddModulatedScanningBeam`. The following can be adjusted in a proton field:

- Proximal target margin `IonBeam.ProximalTargetMargin`.
- Distal target margin `IonBeam.DistalTargetMargin`.
- Lateral margins `IonBeam.ProximalTargetMargin`.

It is now possible to optimize and calculate a proton plan:

- `IonPlanSetup.CalculateDVHEstimates` for calculating DVH Estimates and generating optimization objectives based an a proton DVH Estimation model.
- `IonPlanSetup.CalculateBeamLine` for calculating the beamline.
- `IonPlanSetup.OptimizeIMPT` for optimizing a proton modulated scanning plan.

Additionally, the proton plan has the following new methods:

- `IonPlanSetup.SetNormalization` for modifying Plan normalization mode.
- `IonPlanSetup.SetOptimizationMode` for changing plan normalization mode (Multi-field/single field optimization).

- `IonPlanSetup.PatientSupportDevice` for retrieving details of the patient support device.

It is now possible to copy a plan using `Course.CopyPlanSetup` to create a plan in another image that is not registered.

Structure Modeling

It is now possible to use long structure identifiers (64 characters).

New method `StructureSet.AddStructure(StructureCodeInfo)` allows creation of structures based on the structure code. The DICOM structure type and structure identifier are automatically assigned.

Linkage between reference points and structures has been removed. Consequently, the `API PlanSetup.AddReferencePoint`, which creates a reference point and links it to the plan, no longer takes a structure as parameter.

Method `ReferencePoint.PatientVolumeId` has been removed. Use method `ReferencePoint.Id` instead.

It is now possible to enumerate patient reference points using `Patient.ReferencePoints`.

It is now possible to create new patient reference points with `Patient.AddReferencePoint`.

BrachyVision

There are the following new getter properties for brachy applicator group number: `Catheter.GroupNumber`, `BrachySolidApplicator.GroupNumber`.

General Improvements

The following improvements have also been made to ESAPI:

- New setter properties for names, IDs, and Comments: `Course.Comment`, `PlanSetup.Name`, `PlanSetup.Comment`, `PlanSum.Name`, `Beam.Id`, `Beam.Name`, `Beam.Comment`, `Structure.Name`, `Structure.Comment`, `StructureSet.Name`, `StructureSet.Comment`, `ReferencePoint.Id`.
- The following functionalities are now available in the clinical environment:`Course.RemovePlanSetup` and `Patient.RemoveCourse`. Additionally, the `SearchBodyParameters` class and all its properties are now available in the clinical environment.
- You can now change the status of a Retired script to Approved.
- The following object types are now displayed in Scope Window after modifications are done by the script: `MImage`, `MBeam`, `MSlice`, `MPlanSum`.
- A new dialog is added to show the list of modified objects after a script execution.

- A new check box is added to the script manager window to choose whether the modified objects are always displayed after running a script. If not selected, the modifications dialog will be displayed only if a script is approved for evaluation.

Visual Scripting

Visual scripting now contains a new menu item: **Export/Create ESAPI project**. It is now possible to convert Visual script into Visual Studio ESAPI project, which can be edited/compiled and run as Eclipse binary plugin. Due to that:

- The Code tab has been removed.
- `PlanSetup Clinical Goals` added in ESAPI 16.0 are accessible also in Visual Scripting.
- The `DVH Metrics` element now works with `PlanSum`.

Supported Script Types

Eclipse supports the script types listed below.

Plug-ins

Plug-ins are launched from the Eclipse user interface. After the launch, the plug-in gains access to the data of the currently open patient.

Eclipse supports two types of plug-ins:

- A single-file plug-in: A source code file that Eclipse reads, compiles on the fly, and connects to the data model of the running Eclipse instance.
- A binary plug-in: A compiled .NET assembly that Eclipse loads and connects to the data model of the running Eclipse instance.

Eclipse creates a Windows Presentation Foundation child window that the script code can then fill in with its own user interface components. The plug-in scripts receive the current context of the running Eclipse instance as an input parameter. The context contains the patient, plan, and image that are active in Eclipse when the script is launched. The plug-in scripts work only for one patient at a time in Eclipse.

Executable Applications

A stand-alone executable is a .NET application that references the Eclipse Scripting API class library. It can be launched just like any Windows application.

Stand-alone executables can be either command-line applications, or they can leverage any .NET user interface technology available on the Windows platform.

While the plug-in scripts are restricted to work for one single patient opened in Eclipse, the stand-alone executable can scan the database and open any patient.

Read-Only and Write-Enabled Scripts

Binary plug-ins and executables can be either read-only or write-enabled. Read-only scripts have only read access to the treatment planning data of Eclipse. Write-enabled scripts can be used for changing Eclipse objects.

Visual Scripts

Visual scripts are created in Visual Scripting Workbench and executed in Eclipse as single-file plug-in scripts. They can be either read-only or write-enabled. Visual scripts are launched from the Visual Scripting Workbench, or from Eclipse in the same way as ordinary single-file plug-in scripts.

Eclipse Scripting API Object Model

The Eclipse data model is presented in the Eclipse Scripting API as a collection of .NET classes with properties and methods. The class hierarchy is an abstraction over the ARIA Radiation Therapy Management (RTM) data model and uses similar terminology as the DICOM object model.

The classes of the object model hide all the details of interacting with the database and creating the in-memory representations of the Eclipse data. Because the Scripting API is a .NET class library, all details of managing the memory and other low-level resources are also transparent to you when you create scripts.

Eclipse Scripting API Concepts

The most important concepts of the Eclipse Scripting API are described below.

Coordinate System and Units of Measurement

The Eclipse Scripting API uses the following coordinate systems and units of measurement.

Distances and Positions

In all methods and properties that work with distances and positions, the unit of measurement is millimeters. The positions in 3D space are returned using the DICOM coordinate system. Note that this differs from the Planning Coordinate system used in the Eclipse user interface, where the unit of measurement is centimeters. In addition, when the coordinate values are displayed in the Eclipse user interface, the following are taken into account:

- The possible user-defined origin of an image.
- The treatment orientation of the plan.
- The axis definition of the planning coordinate system.

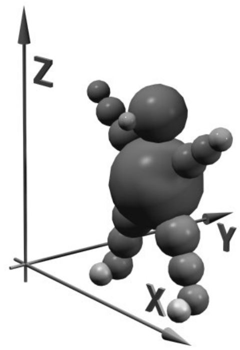


Figure 1 DICOM Coordinate System

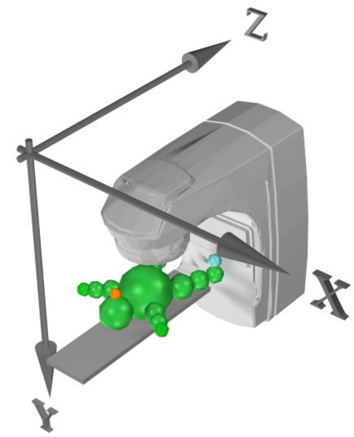


Figure 2 Standard Planning Coordinate System

The Eclipse Scripting API has methods that convert values from the DICOM coordinate system to the same representation that is used in the Eclipse user interface.

For more information on the display of 3D coordinates in the Eclipse user interface, refer to *Eclipse Photon and Electron Reference Guide*.

For more information on the DICOM coordinate system, refer to the DICOM standard.

DOSE VALUES

In the Eclipse Scripting API, dose values are always represented with the separate `VMS.TPS.Common.Model.Types.DoseValue type`. In addition to the actual floating point value of the variable, this type also holds the measurement unit of the dose. The measurement unit can be Gy or cGy, depending on the selected clinical configuration. It can also be a percentage if relative dose is used.

TREATMENT UNIT SCALES

All methods and properties of the Eclipse Scripting API return the treatment unit and accessory properties in the IEC61217 scale. This feature allows you to create scripts despite the scale interpretation differences between treatment unit vendors.

User Rights and HIPAA

The Eclipse Scripting API uses the same user rights and HIPAA logging features as Eclipse. When a plug-in script is executed, the script applies the same user rights as were used to log into Eclipse.

When you execute a stand-alone executable script, the user name and password are automatically passed via the new single sign-on technology implemented in the Eclipse release so that no additional dialogs are required to authenticate the user to the system.

According to HIPAA rules, a log entry is made for each patient opened by a standalone script. Additionally, the Eclipse Scripting API follows the rules of department categorization of ARIA RTM.

Working with Several Patients

The context of the running Eclipse instance is passed to plug-in scripts. They work only for the one patient that is selected in that context. In contrast, stand-alone executables can open any patient in the database. However, only the object model of a single patient is available at a time. The previous patient data must be explicitly closed before another patient is opened. If you try to access the data of a patient that has been closed, an access violation exception is generated.

Overview of the Object Model

The following diagram gives an overview of the Image-related objects in the Eclipse Scripting API.

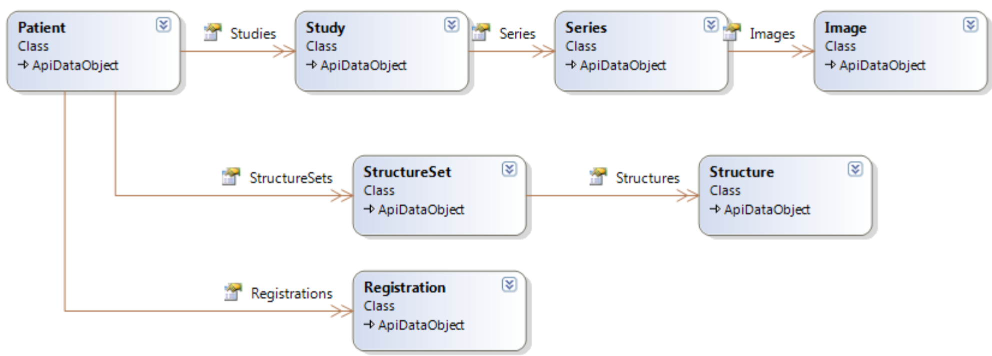


Figure 3 Image Data Model

The diagram contains the following objects:

- A Patient that has a collection of Study, StructureSet and Registration objects.
- A Study that has a collection of Series objects.
- A Series that has a collection of Image objects.
- A StructureSet that has a collection of Structure objects.

Another important section of the Eclipse Scripting API is the model of Plan-related objects shown in the following diagram.

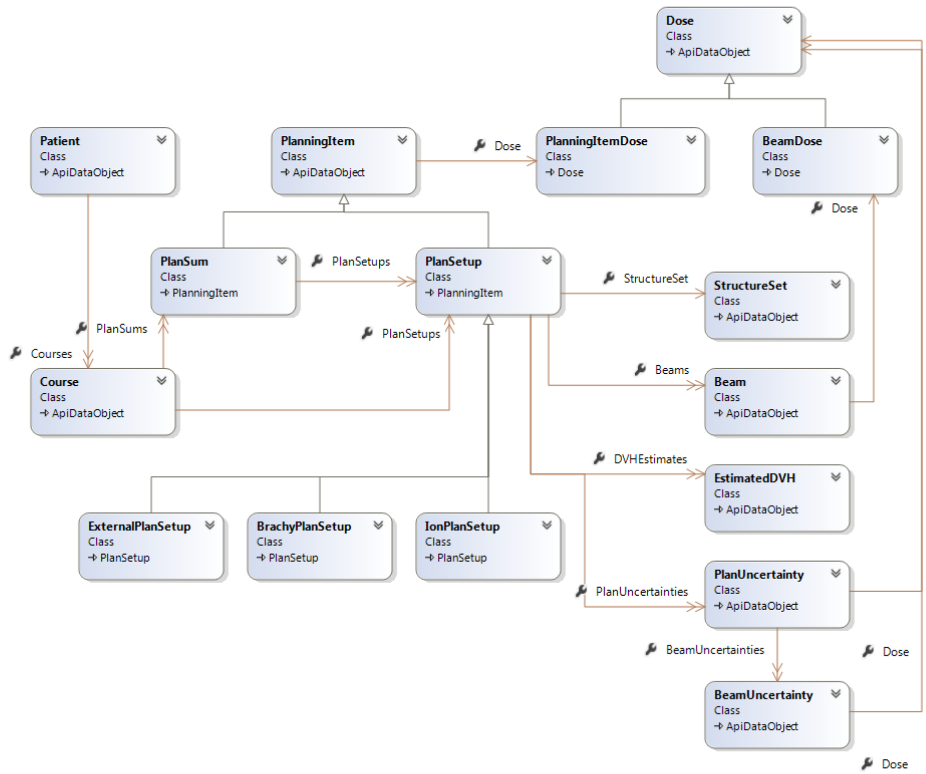


Figure 4 Plan Data Model

The diagram contains the following objects:

- A Patient that has a collection of Course objects.
- A Course that has a collection of PlanSetup and PlanSum objects. Each of them is derived from the common PlanningItem base class. Each PlanSetup object is an ExternalPlanSetup, a BrachyPlanSetup, or an IonPlanSetup.
- A PlanningItem class that has a direct (but nullable) relationship with a PlanningItemDose class.
- A PlanSetup that has a collection of Beam objects. Beam has a direct (but nullable) relationship with a BeamDose class.
- A PlanSetup that has a direct (but nullable) relationship with StructureSet and EstimatedDVH objects.
- A PlanSetup that has a collection of PlanUncertainty objects.
- A PlanUncertainty has a collection of BeamUncertainty objects, and a direct (but nullable) relationship with a Dose class.
- BeamUncertainty has a direct (but nullable) relationship with a Dose class.

The object model related to Plan optimization is visualized in Figure 5.

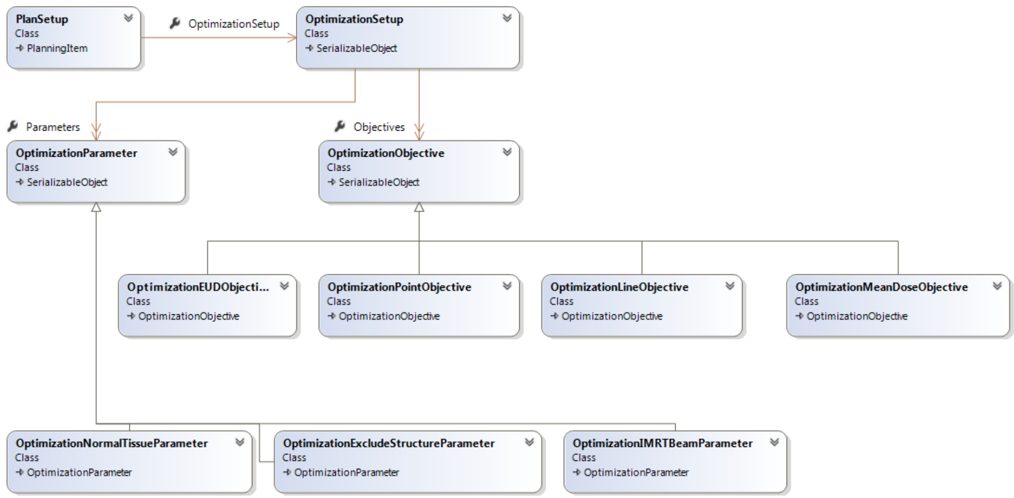


Figure 5 Plan Optimization Data Model

The diagram contains the following objects:

- A PlanSetup that has an association to the OptimizationSetup.
- An OptimizationSetup that has a collection of OptimizationParameter objects. Each OptimizationParameter object is an OptimizationNormalTissueParameter, OptimizationExcludeStructureParameter, OptimizationIMRTBeamParameter, or OptimizationPointCloudParameter.
- An OptimizationSetup that has a collection of OptimizationObjective objects. Each object is an OptimizationPointObjective, OptimizationEUDObjective, OptimizationLineObjective, or OptimizationMeanDoseObjective.

The following diagram shows the objects related to an individual Beam:

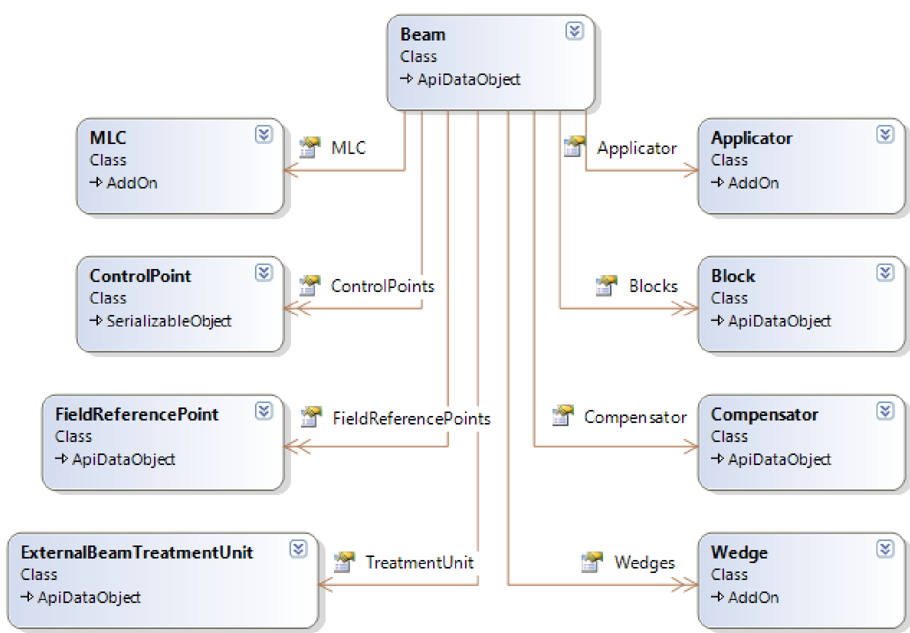


Figure 6 Beam Data Model

The diagram contains the following objects:

- An MLC and a ControlPoint collection of the Beam.
- An Applicator, a Compensator and a collection of Blocks and Wedges if defined for the Beam.
- A collection of FieldReferencePoint objects for the Beam.
- An ExternalBeamTreatmentUnit object that represent the treatment unit.

The following diagram shows the data model for brachytherapy plans:

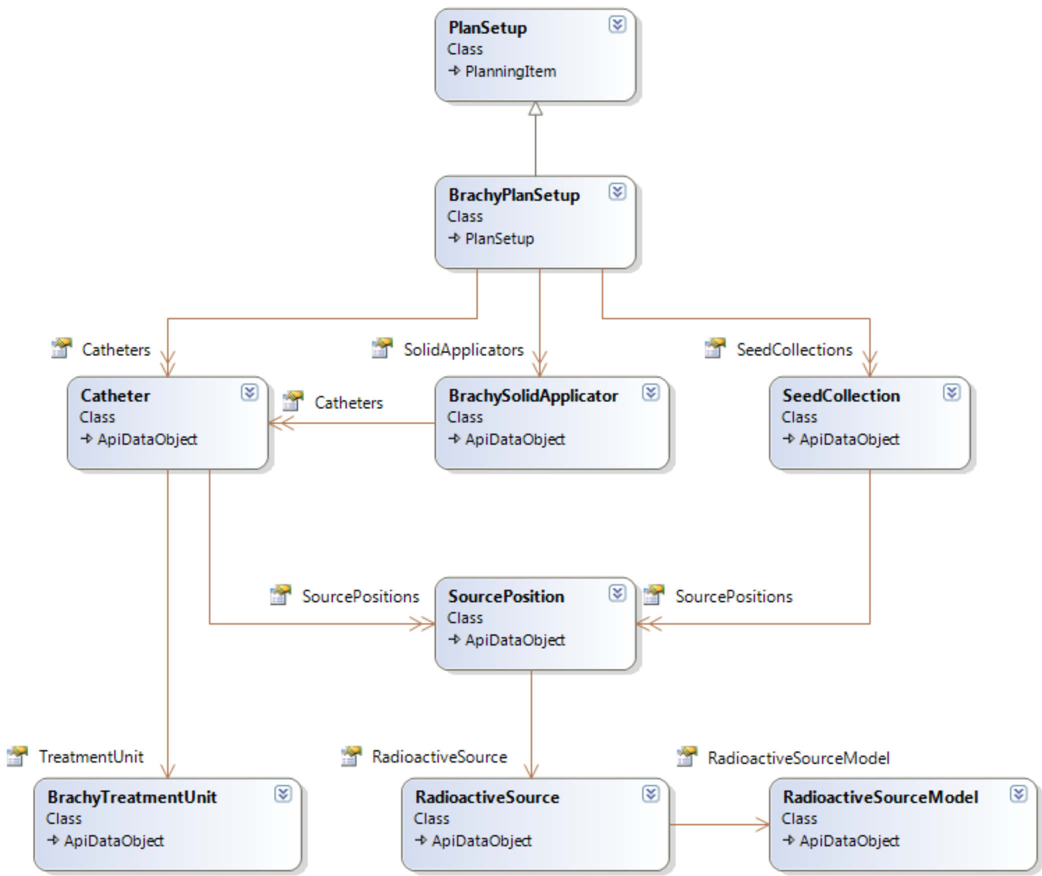


Figure 7 Brachytherapy Data Model

The diagram contains the following objects:

- A BrachyPlanSetup is derived from PlanSetup. The BrachyPlanSetup has a collection of Catheters, BrachySolidApplicators, and SeedCollections. Note that BrachyPlanSetups can be accessed through the Course in the same way as PlanSetups.
- A BrachySolidApplicator has a collection of Catheters.
- A Catheter (applicator channel central line or needle) has a BrachyTreatmentUnit and a collection of SourcePositions.
- A SeedCollection has a collection of SourcePositions.
- A SourcePosition has a RadioactiveSource.
- A RadioactiveSource has a RadioactiveSourceModel.

The following diagram details the proton plan data model:

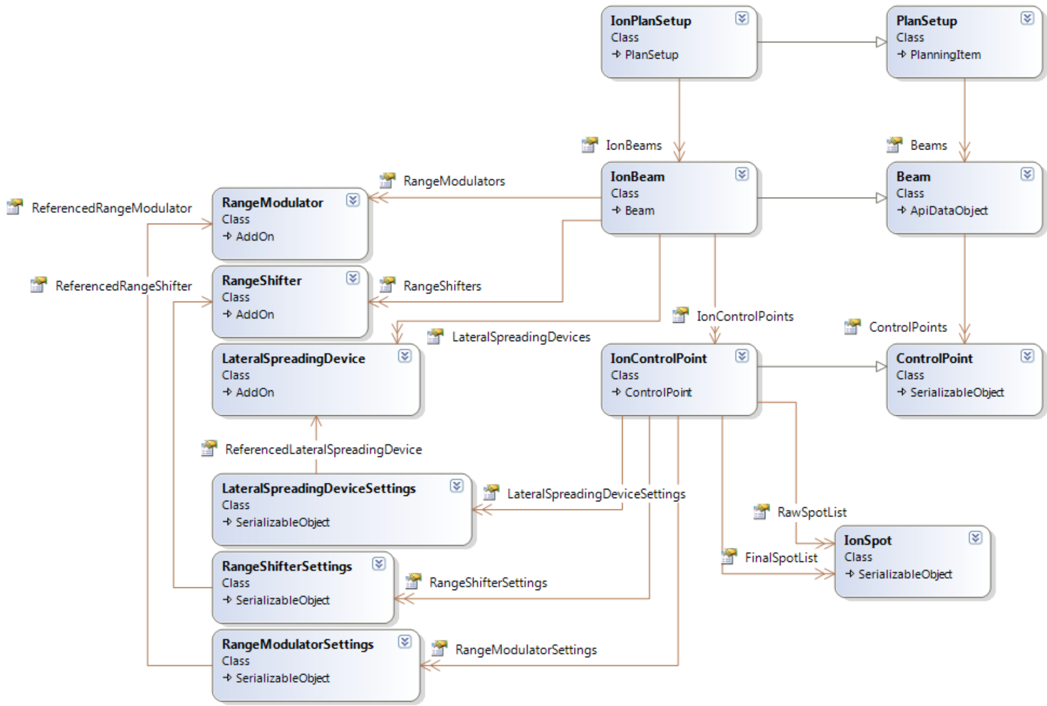


Figure 8 Proton Plan Data Model

The diagram contains the following objects:

- An IonPlanSetup is derived from PlanSetup. The IonPlanSetup has a collection of IonBeams.
- IonBeam is derived from Beam. The IonBeam has collections of IonControlPoints, RangeModulators, RangeShifters, and LateralSpreadingDevices.
- The IonControlPoint is derived from ControlPoint. It provides access to the raw spot list IonSpot objects through property RawSpotList and access to the final spot list through property FinalSpotList.
- IonControlPoint has collections of LateralSpreadingDeviceSettings, RangeShifterSettings, and RangeModulatorSettings.
- LateralSpreadingDeviceSettings contains control-point-level settings for a LateralSpreadingDevice owned by the IonBeam. RangeShifterSettings contains control-point-level settings for a RangeShifter owned by the IonBeam. RangeModulatorSettings contains control-point-level settings for a RangeModulator object owned by the IonBeam.

The properties of each object are described in detail in *Eclipse Scripting API Online Help*.

Installing The Eclipse Scripting API in a Developer Environment

You can install Eclipse Scripting API libraries and components in a separate Windows developer environment with the Eclipse Scripting API installer. The ESAPI installer installs the Eclipse Script Wizard, *Eclipse Scripting API Online Help*, and the DLL files needed for creating and compiling scripts. Once these components are installed, you can create and compile standalone and plug-in scripts without having the Eclipse treatment planning system installed. Executing scripts in this developer environment requires Eclipse to be installed.

Note: Do not use this installer for updating the installed Eclipse Scripting API on any clinical system. Varian prohibits the use of the installer for this purpose. Only authorized Varian service personnel is allowed to change the installation on clinical systems.

Note: ESAPI scripts only run on computers that have Eclipse installed.

Installing the Eclipse Scripting API in a developer environment has the following benefits:

- Provides easier access to the Eclipse Script Wizard and *Eclipse Scripting API Online Help*.
- Allows you to preview new ESAPI releases.
- Allows you to create and compile scripts on workstations that do not have Eclipse installed (as is the case with local workstations in Citrix environments).

Install The Eclipse Scripting API

1. Save the installer (*Varian_Eclipse_Scripting.msi*) on your local computer.
2. To start the installation process, double click *Varian_Eclipse_Scripting.msi*.
3. Click **Next** and follow the instructions in the wizard to complete the installation.

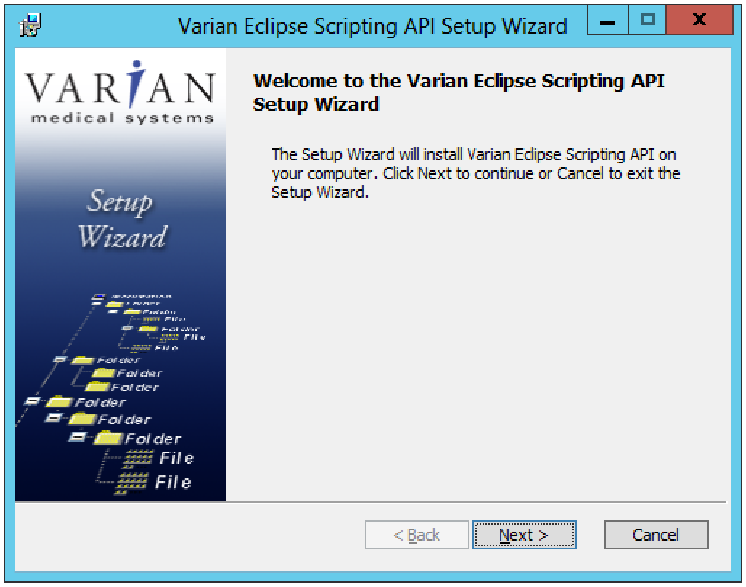


Figure 9 Eclipse Scripting API Setup Wizard

A folder called Varian is added to the desktop and to the Windows **Start** menu.

This folder contains the subfolder Eclipse Scripting API, which includes a shortcut to the **Eclipse Script Wizard** and to *Eclipse Scripting API Online Help*:

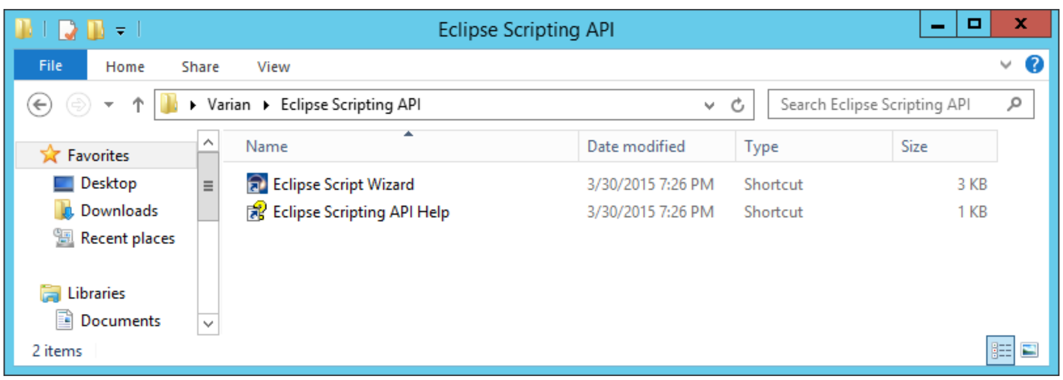


Figure 10 Folder Structure of the Installed Eclipse Scripting API

The installer also adds libraries needed for running the **Eclipse Script Wizard** and for compiling ESAPI plug-in and stand-alone executable scripts.

The **Eclipse Script Wizard** creates Visual Studio project files that reference these libraries so that ESAPI projects can be compiled on the developer workstation where the Eclipse Scripting API is installed.

Getting Started with The Eclipse Scripting API

To get quickly started with the Eclipse Scripting API, you can:

1. Copy the code shown below to a file.
2. Save the file with a .cs extension on the hard disk of your workstation.

```
using System;
using System.Text;
using System.Windows;
using VMS.TPS.Common.Model.API;

namespace VMS.TPS
{
  class Script
  {
    public Script()
    {
    }
    public void Execute(ScriptContext context)
    {
      if (context.Patient != null)
      {
        MessageBox.Show("Patient id is " + context.Patient.Id);
      }
      else
      {
        MessageBox.Show("No patient selected");
      }
    }
  }
}
```

Figure 11 Sample Script Code

3. In Eclipse, select Tools > Scripts.
4. Select the Directory: [path_to_your_own_scripts] option.
5. To locate the script that you created, click Change Directory.
6. In the Scripts dialog box, select the script from the list and click Run. The script displays a message box which contains the ID of the patient that is open in Eclipse.

Using Example Scripts

The Eclipse Scripting API includes example scripts for a few of the supported script types. You can first copy the example scripts by using the Script Wizard, and then compile them by using Visual Studio.

If you do not have Visual Studio available, you can compile the examples with the MSBuild program, which is included in the Microsoft .NET framework.

Copy Example Scripts

To copy the example scripts to your own location:

1. From the Start menu, select **Varian > Eclipse Scripting API > Eclipse Script Wizard**.
2. Click the **Copy Example Scripts** tab.
3. To select a location for copying the example scripts, click **Browse**.
4. Click **Copy**. The example scripts are copied to the specified location.

Compile Example Scripts

To compile the examples by using Visual Studio:

1. Open the Visual Studio project files.
2. Compile the examples.

After this, you can launch the example scripts.

If you do not have Visual Studio available, you can compile the examples with the MSBuild program, which is included in the Microsoft .NET framework and the Microsoft Build Tools package.

To compile the examples by using MSBuild:

1. In the file browser, go to the directory where you copied the example scripts.
2. Open Command Prompt.
3. Enter the following information on the command line:
   - The path to the directory where MSBuild.exe is located.
   - The name of the project file.
   - Platform specification for x64.

   For example:

   ```
   C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe
   Example_DVH.csproj /p:Platform=x64
   ```
4. To compile the example, press ENTER.

Creating Scripts

You can create scripts manually or by using the Script Wizard.

Creating Plug-in Scripts

The following sections give you step-by-step instructions on creating different types of plug-in scripts supported by the Eclipse Scripting API.

Create a Single-File Plug-in with the Script Wizard

To create a single-file plug-in with the Script Wizard, follow these guidelines:

1. From the **Start** menu, select **Varian > Eclipse Scripting API > Eclipse Script Wizard**.
2. Enter a name for the new script.
3. Select the **Single-file plug-in** option.
4. To select the location for storing the script, click **Browse**. By default, the script is stored in the user-specific Documents folder.
5. Click **Create**.
6. The Script Wizard creates the following folders in the location that you selected:
   - *Project* folder: Contains a script-specific sub-folder where the Microsoft Visual Studio project file is stored.
   - *Plugins* folder: Contains the source code file for the single-file plug-in.

   The Script Wizard launches Visual Studio.
7. Edit the source code file according to your needs. You can use Visual Studio and its IntelliSense support for editing the file, but they are not required.
8. You do not have to compile the plug-in, because Eclipse compiles it automatically on the fly.

Create a Binary Plug-in with the Script Wizard

To create a binary plug-in with the Script Wizard, follow these guidelines:

1. From the **Start** menu, select **Varian > Eclipse Scripting API > Eclipse Script Wizard**.
2. Enter a name for the new script.
3. Select the **Binary plug-in** option.
4. To select the location for storing the script, click **Browse**. By default, the script is stored in the user-specific *Documents* folder.
5. Click **Create**.

   The Script Wizard creates the following folders in the location that you selected:

- *Project* folder: Contains a script-specific subfolder where the Microsoft Visual Studio project file and source code file are stored.
- *Plugins* folder: Contains the compiled plug-in dlls. From this folder, the dll can be loaded into Eclipse.

The Script Wizard launches Visual Studio.

6. Edit the source code file according to your needs.
7. Compile the plug-in, for example, by using Visual Studio. The resulting plug-in dll is saved into the Plugins folder. Note that you can also use the MSBuild tool to compile the binary plug-in. For an example, see Chapter *"Getting Started with The Eclipse Scripting API",* section *"Compile Example Scripts"*. For more information about MSBuild, refer to Microsoft documentation.

Create a Single-File Plug-in Manually

If you want to create a single-file plug-in without the Script Wizard, follow these guidelines. For an example of a source code file, see Chapter *Getting Started with The Eclipse Scripting API*.

1. Create an empty C# source code file.
2. Add the using statements for the `System` and `System.Windows` namespaces.
3. Add the using statements for the following namespaces:
   - `VMS.TPS.Common.Model.API`
   - `VMS.TPS.Common.Model.Types`
4. Add a namespace called `VMS.TPS`.
5. To the `VMS.TPS` namespace, add a public class called `Script`.
6. To the `Script` class, add a constructor without parameters, and a method called `Execute`.
7. Define the return type of the `Execute` method as void.
8. To the '`Execute`' method, add the following parameters:
   - The context of the running Eclipse instance. The parameter type is `VMS.TPS.Common.Model.API.ScriptContext`.
   - A reference to the child window that Eclipse creates for the user interface components (optional). The parameter type is `System.Windows.Window`.
9. You do not have to compile the plug-in, because Eclipse compiles it automatically on the fly.

Create a Binary Plug-in Manually

If you want to create a binary plug-in without the Script Wizard, follow these guidelines:

1. In Microsoft Visual Studio, create a new Class Library project. Select x64 as the Solution Platform.
2. Create the source code in the same way as for a single-file plug-in. For instructions, see Chapter *"Creating Scripts",* Section *"Create a Single-File Plug-in Manually"*.

3. Use the following file name extension for the dll: .esapi.dll. In this way, Eclipse recognizes the plug-in and can load it.
4. Add references to the following class libraries of the Eclipse Scripting API:
   - `VMS.TPS.Common.Model.API.dll`
   - `VMS.TPS.Common.Model.Types.dll.`

   On the basis of this information, the dll can access the Eclipse Scripting API. The assemblies are located under the installation directory of the Eclipse Scripting API, in the API subdirectory.
5. Compile the plug-in into a .NET assembly (a dll), for example, by using Visual Studio.

For more information on how to create a .NET assembly and add references to class libraries, refer to Microsoft documentation.

Storing Plug-in Scripts

If you want to make the created scripts available for all workstations, store them into the *System Scripts* directory. The *System Scripts* directory is a shared directory on the Varian System server.

You can access the *System Scripts* directory by clicking the **Open Directory** button in the Scripts dialog box.

Creating Stand-Alone Executable Applications

The following sections give you step-by-step instructions on creating stand-alone executables supported by the Eclipse Scripting API.

Create a Stand-Alone Executable with the Script Wizard

To create a stand-alone executable with the Script Wizard, follow these guidelines:

1. From the Start menu, select Varian > Eclipse Scripting API > Eclipse Script Wizard.
2. Enter a name for the new script.
3. Select the Standalone executable option.
4. To select the location for storing the script, click Browse.
5. Click Create.
6. The Script Wizard creates a Projects folder in the location that you selected. The folder contains a script-specific subfolder where the Microsoft Visual Studio project file and source code file are stored. The Script Wizard launches Visual Studio.
7. Edit the source code file according to your needs.

Create a Stand-Alone Executable Manually

If you want to create stand-alone executables without the Script Wizard, follow these guidelines:

8. In Microsoft Visual Studio, create a new project file for the executable. Select x64 as the Solution Platform.
9. Add references to the following class libraries of the Eclipse Scripting API:
   - VMS.TPS.Common.Model.API.dll
   - VMS.TPS.Common.Model.Types.dll.

   On the basis of this information, the executable can access the Eclipse Scripting API. The assemblies are located under the installation directory of the Eclipse Scripting API, in the API subdirectory.
10. In the main method of the executable file, use the static CreateApplication method to create an instance of the VMS.TPS.Common.Model.API.Application class. This class represents the root object of the data model. The CreateApplication method also initializes the Eclipse Scripting API.
11. Dispose of the instance when the stand-alone executable exits to free the unmanaged resources in the Eclipse Scripting API. For more information on disposing of objects, refer to Microsoft documentation of the IDisposable interface.
12. Use a single-threaded apartment (STA) as the COM threading model of the executable. The Eclipse Scripting API must only be accessed from a single thread that runs in the default application domain. For more information about threading and application domains, refer to Microsoft documentation.

The following is the code for a sample stand-alone executable in C# language:

```
using System;
using System.Linq;
using System.Text;
using System.Collections.Generic;
using VMS.TPS.Common.Model.API;
using VMS.TPS.Common.Model.Types;

namespace StandaloneExample
{
  class Program
  {
    [STAThread]
    static void Main(string[] args)
    {
      try
      {
        using (Application app = Application.CreateApplication())
        {
          Execute(app);
        }
      }
      catch (Exception e)
      {
        Console.Error.WriteLine(e.ToString());
      }
    }
    static void Execute(Application app)
    {
      string message =
        "Current user is " + app.CurrentUser.Id + "\n\n" +
        "The number of patients in the database is " +
        app.PatientSummaries.Count() + "\n\n" +
        "Press enter to quit...\n";
      Console.WriteLine(message);
      Console.ReadLine();
    }
  }
}
```

Figure 12 Sample Code for Stand-alone Executable

13. Compile the project. The stand-alone executable is ready to be run.

For more information on creating and compiling .NET applications, refer to Microsoft documentation.

Changing Scripts to Be Write-Enabled

You can create and save write-enabled scripts as described below when your installation includes the Eclipse Automation license.

14. Create a script with the Eclipse Script Wizard as described in the following sections:
    - Create a Binary Plug-in with the Script Wizard

- Create a Stand-alone Executable with the Script Wizard

15. Add the following line of code above the namespace declaration:

```
[assembly: ESAPIScript(IsWriteable = true)]
```

16. Add a call to the `BeginModifications` method of the `Patient` class as shown in the following code example:

```
[assembly: ESAPIScript(IsWriteable = true)]
namespace VMS.TPS
{
  class Script
  {
    public Script()
    {
    }

    public void Execute(ScriptContext context, System.Windows.Window window)
    {
      Patient patient = context.Patient;
      // BeginModifications will throw an exception if the system is not
      // configured for research use or system is a clinical system and
      // the script is not approved.
      patient.BeginModifications();

      // After calling BeginModifications successfully it is possible
      // to modifiy patient data.
      if (patient.CanAddCourse())
      {
        // E.g. the script adds a new course
        Course newCourse = patient.AddCourse();
        // Continue with other changes
      }
    }
  }
}
```

4. Save or discard modifications as follows:

In a stand-alone executable script, use the `Application` class:

- To save the modifications to the database, call the `SaveModifications` method.
- To discard the modifications, close the patient by calling the `ClosePatient` method. You can then open the patient again if the script still needs to access the patient data.

A plug-in script (single-file or binary) modifies the current data context of the Eclipse Treatment Planning application:

- After the script has been executed, save or discard the modifications in the Eclipse Treatment Planning user interface in the same way as any other change.

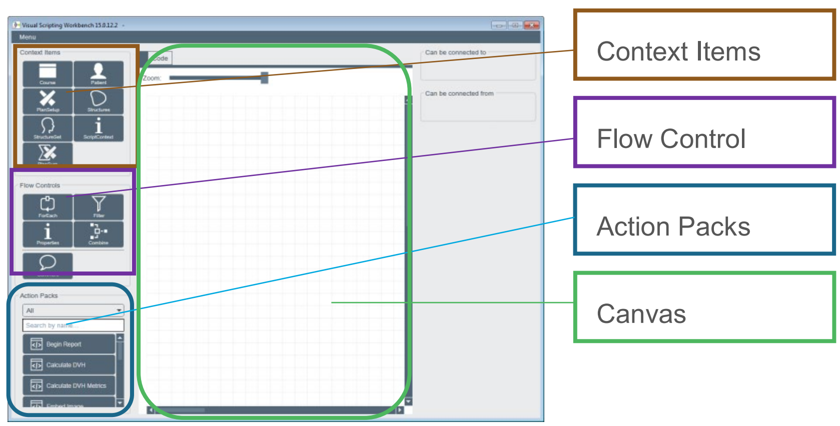
Creating Visual Scripts

Using Visual Scripting Workbench you can create ESAPI scripts with a visual programming method, without the need to know how to program C# code.

Visual Scripting Workbench

The Visual Scripting Workbench is used to create and manage visual scripts. You can create, save, open, delete, export, and immediately run visual scripts in Eclipse. You can also import visual scripts from other users who have exported visual scripts for sharing. Advanced users can generate ESAPI script code with the Visual Scripting Workbench, which they can use as a basis for their own custom ESAPI script. You can open and modify only such scripts in the Visual Scripting Workbench that have been originally created there.

The Visual Scripting Workbench contains three types of script elements—Context Items, Flow Control, and Action Packs. They are dragged to the Canvas to create a script.



Action Packs

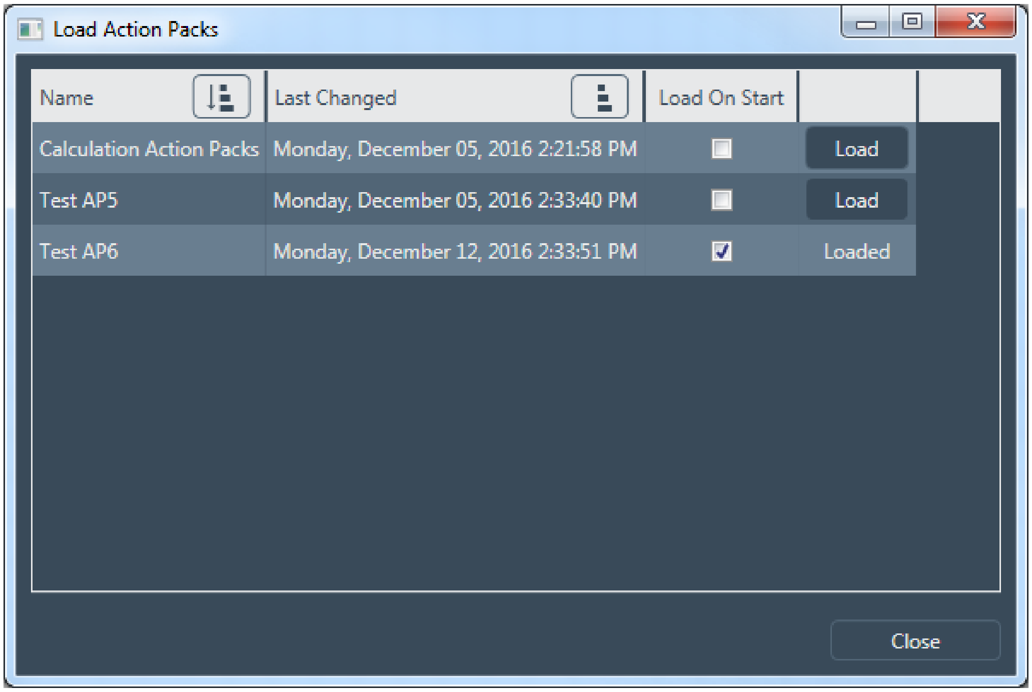
Action packs are modules that perform a single function. They are similar to macros found in other scripting tools. Action packs accept input data, use the input data to perform a function on that data or in Eclipse, and then send output data to the next action pack in line in the flow. The data that flows between action packs are high-level radiotherapy objects like Patients, Plans, and Structures.

Action packs contain also a few helper objects like a Table which is useful in reporting, export, and presenting logical listings of data; and a Report, which represents an electronic report that is used in visual scripting reporting flows.

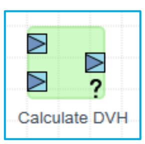
Visual Scripting Workbench contains a number of ready-made action packs, and programmers can also create their own action packs to extend the functionality of Visual

Scripting. The ready-made action packs are all read-only, but customized action packs created by programmers can be write-enabled.

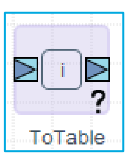
You can manually load custom action packs, or choose which custom action packs to load every time that Visual Scripting is started by choosing the Load Action Packs menu command.



Examples of an action pack:



Calculates the DVH of all input structures.



Formats input information to a table format. Can be used in reporting, export, and in presenting logical listings of data.

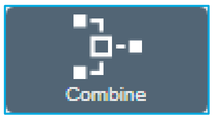
Flow Controls

Using flow controls you can combine action packs into a series of functions. A flow connects the input and output of an action pack. Flow control elements can be used to filter and combine inputs and outputs, and to loop over lists.

The **ForEach** flow control loops over all items in the passed input list and performs the action pack flow contained within it. Note that looping is normally not required in visual scripting flows since most action packs are capable of taking both single context items and lists of items as input.

When inserted between action packs, the **Filter** control filters the context items, such as structures, according to the selected criteria.

The **Properties** control changes the item flowing in a visual script to the selected sub-item. This is used to expose items and properties that are present in the Eclipse Scripting API but are not otherwise exposed by context items.

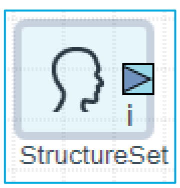


You can use the **Combine** control to combine multiple lists into a single list for further processing in a visual scripting flow.

The **Comment** control is used for adding documentation to visual scripts.

Context Items

Context items pass active Eclipse data, such as plan and structure data, to a flow. Context items may include sub-items (for example, PlanSetup may include StructureSets) that you can also use in a flow.



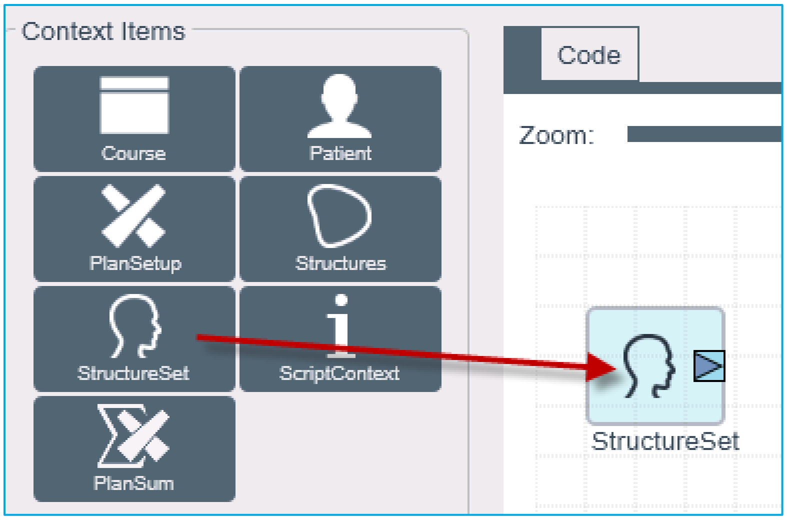
Retrieves the structure set information from the currently active plan.

Canvas

Canvas is an area in the user interface, to which you drag the selected script elements, action packs, context items, and flow controls, to form flows. Use the following functions to select the elements and create connections between them.

Drag a script element to the canvas

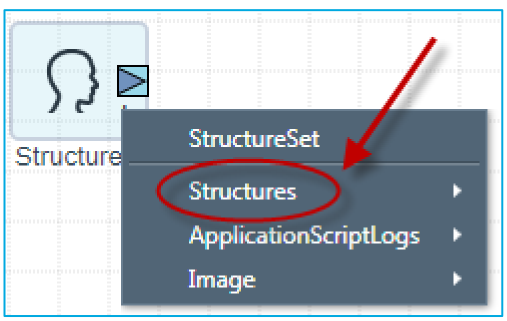
Select an element and drag it to canvas:



Open the menu of a context item

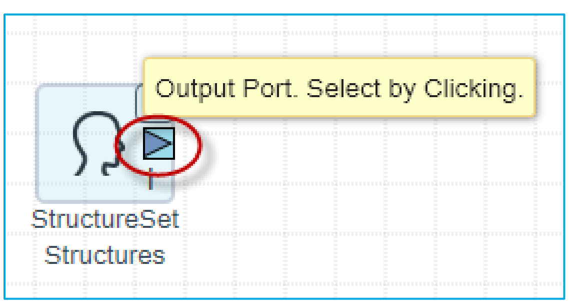
Click the **i** icon in the item.

Select a sub-item, if you wish:

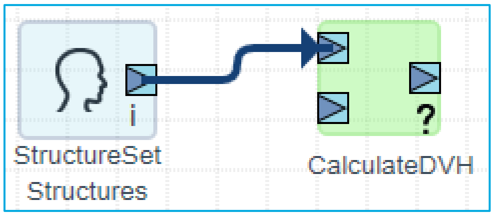


Create a connection between elements

Each context item contains an Output Port to connect the item into other objects. To draw a connection, click the Output Port of the context item:



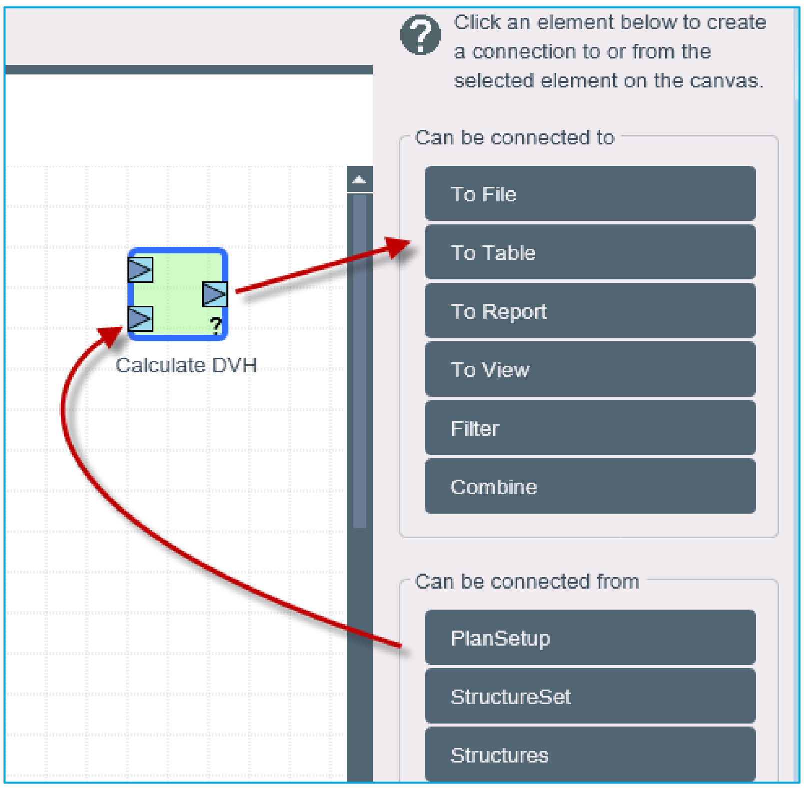
Each action pack contains Input Ports and Output Ports to connect the action packs into other elements. Drag an Input Port to an Output Port to make a connection.



If the line turns orange, the created connection is invalid, or, the validity of the connection depends on the previous input elements in the flow. You can remove an invalid connection by hovering over the middle part of the line, and clicking the X button.

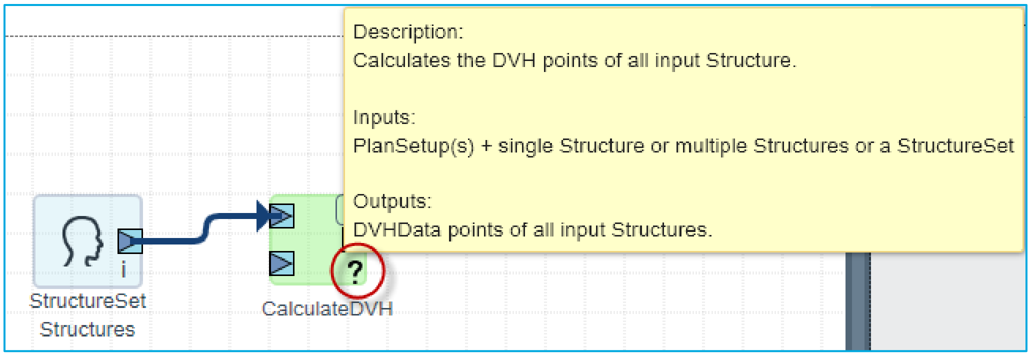
Use the connection pane for creating the connection

The connection pane shows the script elements that can be used as inputs or outputs of the selected element:



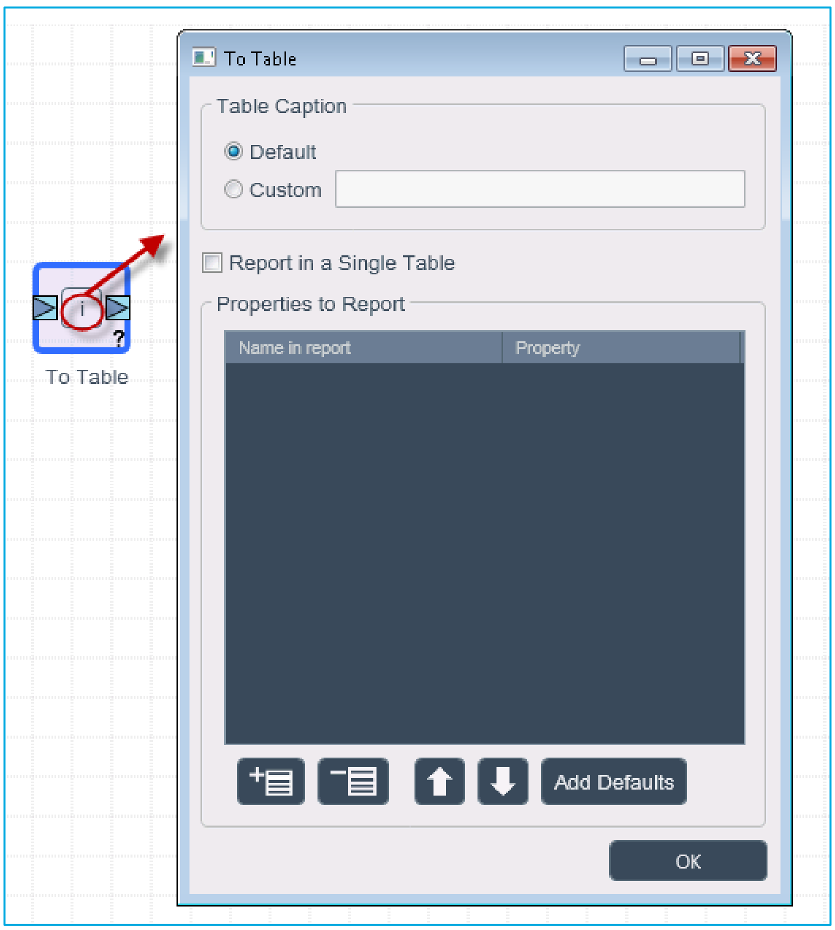
View accepted inputs and outputs of an action pack

Hover over the question mark in an action pack to view what kind of inputs and outputs it accepts:



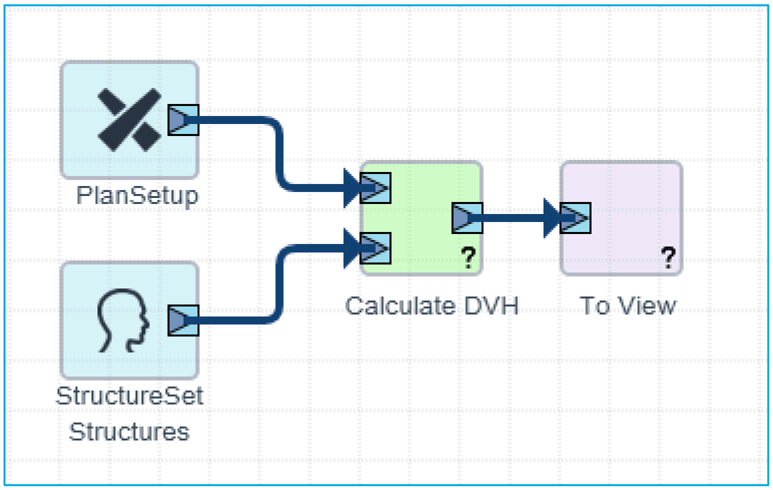
View action pack settings

Some action packs may need additional settings configured in order to properly execute in a visual script. For example, the ToFile action pack needs to know the path to the file where data will be saved. You can view and edit action pack settings by clicking the **i** in the element.



Example—Visual Script for Calculating DVHs

The simple visual script below calculates dose volume histograms (DVHs) for all structures of the active patient and displays them. The **CalculateDVH** action pack has two inputs and one output. Its function is to calculate the DVH for all structures passed to it for the specified plan and output those DVHs. Inputs to this action pack are the active **PlanSetup** and the active structure set (**StructureSet → Structures**) in Eclipse. The output is a DVH object, which is the input of the **ToView** action pack.



Create and Test a Visual Script

You can access the Visual Scripting Workbench in Eclipse **External Beam Planning** and **BrachyVision**. To create a visual script:

1. In Eclipse, select **Tools > Visual Scripting**. The main window of the Visual Scripting Workbench appears.
2. Select an action pack to start your flow and drag it onto the canvas.
3. Add other necessary action packs and context items to the canvas and connect them by using the following options:
    - Click the **Output Port** triangle in an element. The application suggests an element to which to connect by showing a dotted line to the **Input Port** of the element. To confirm the connection, click the triangle again. To remove the connection, hover over the middle part of the line, and click the X button.
    - Click the element and choose another element to which you want to connect in the upper right corner under **Can be connected to** or **Can be connected from**.

    For some of the action packs and context items, you can further define what kind of information is retrieved and how. To view additional options, click the i button in the element. To remove an added element from the canvas, hover over the right upper corner of the element, and click the X button.
4. Use flow controls as an aid if you wish.
5. When all inputs have been defined, test the visual script in Eclipse.
6. Go to **Menu > Save and Execute** in Eclipse.

Run a Visual Script

To run a visual script in Eclipse, choose **Menu > Save and Execute** in Eclipse.

Save a Visual Script

To save a visual script, choose **Menu > Save**, or **Menu > Save As**. The script is stored in a user-specific folder on the server.

Add a Visual Script as a Favorite

To add the script to the Eclipse Tools menu as a favorite, choose **Menu > Add to Favorites**.

To remove a script from the Tools menu, choose **Menu > Delete from Favorites**.

Export and Import a Visual Script

To export a script, choose **Menu > Export**.

To import a script, choose **Menu > Import**.

Create or Delete Scripts

To close the current script and create another script, choose **Menu > New Script**.

To delete a script, choose **Menu > Delete**.

Example—Create a Visual Script for Calculating DVHs

To create the example visual script illustrated in Example—Visual Script for Calculating DVHs:

7. In Visual Scripting Workbench, drag the **CalculateDVH** action pack onto the canvas.
8. While the **CalculateDVH** action pack is selected, click on the **ToView** action pack in the **Can be connected** to section on the top right.
9. Select the **CalculateDVH** action pack again, and click the **PlanSetup** context item in the **Can be connected from** section on the middle right.
10. Click the **StructureSet > Structures** context item to finish the flow.

Example Visual Scripting Flows

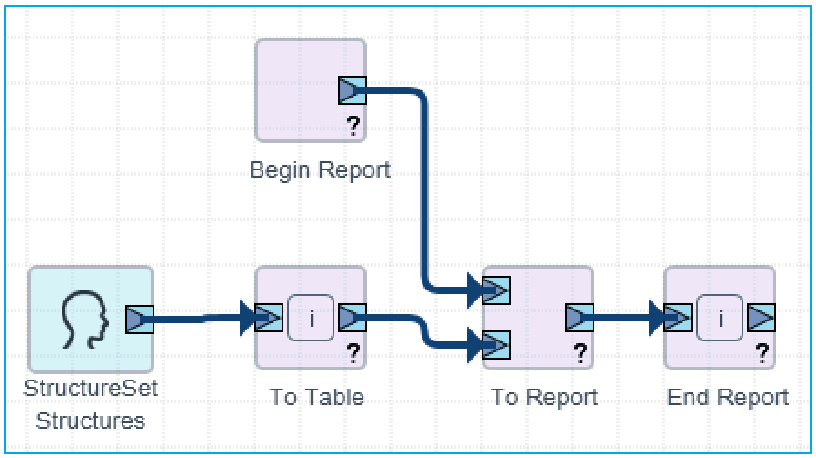
This section describes how to achieve a few typical activities with visual scripting.

Create a Custom Treatment Planning Report

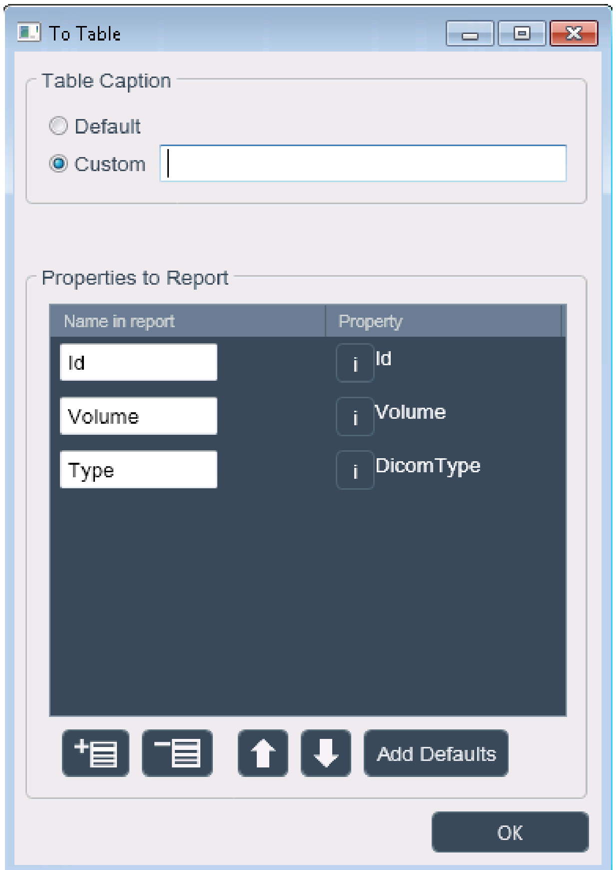
You can create customized treatment planning reports that display only the selected properties, for example, of a structure set.

The following flow creates a PDF report that includes a table with the ID, volume, and type information of all structures. The PDF is created when the script is executed in Eclipse.

1. In Visual Scripting Workbench, drag the **BeginReport** action pack onto the canvas.
2. Add reporting elements as desired. For example, to report on the properties of loaded structures, send **StructureSet → Structures** context item to the **ToTable** action pack and flow that to the **ToReport** action pack. Finish the flow with **EndReport**.



3. Configure the **ToTable** action pack to select the desired properties from the structures and put them in a table. In this case, select the structure ID, the type, and the volume.

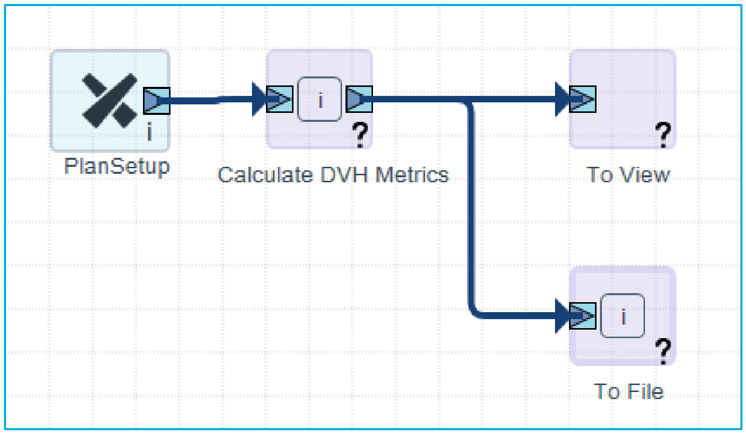


Evaluate DVH Metrics

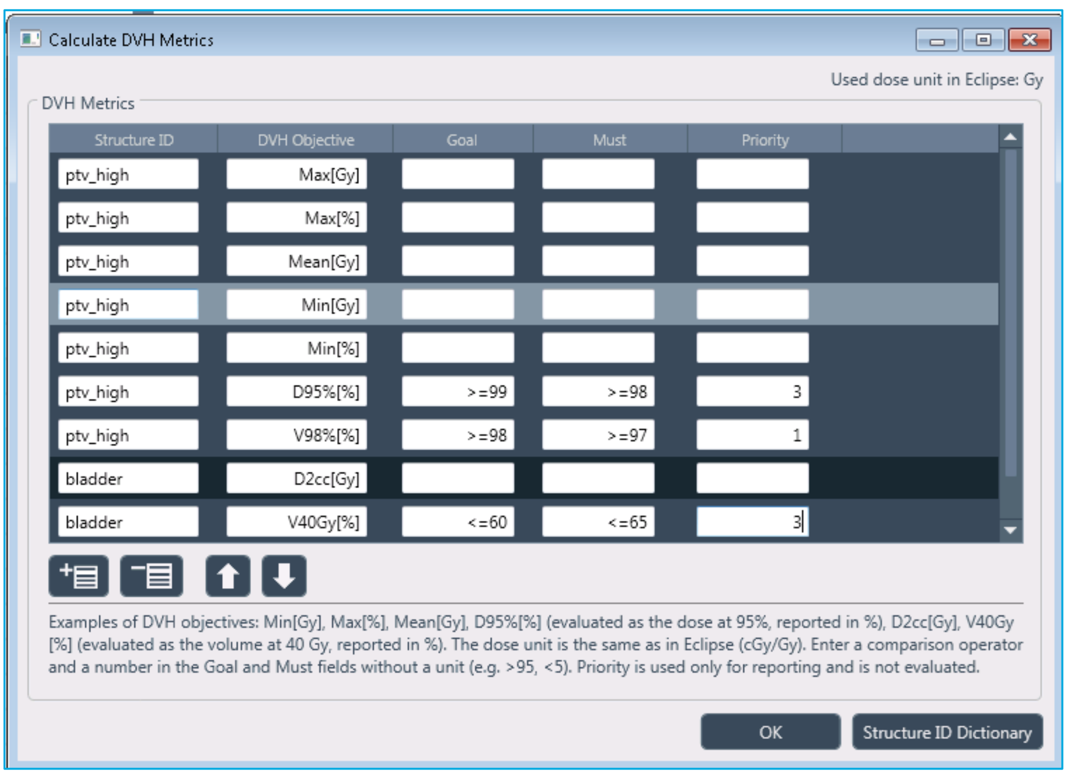
You can evaluate customized DVH metrics using the syntax defined by Mayo, et al. in Establishment of practice standards in nomenclature and prescription to enable construction of software and databases for knowledge-based practice review (Pract Radiat Oncol. 2016 Jul-Aug;6(4):e117–26. doi: 10.1016/j.prro.2015.11.001. Epub 2016 Jan 26).

The following flow evaluates user-defined metrics, exports them to a CSV file, and also shows the metrics to the user in a table.

4. In Visual Scripting Workbench, drag the **Calculate DVH Metrics** action pack onto the canvas.
5. Add the **ToView** action pack and connect it.
6. Add the **ToFile** action pack and connect it.

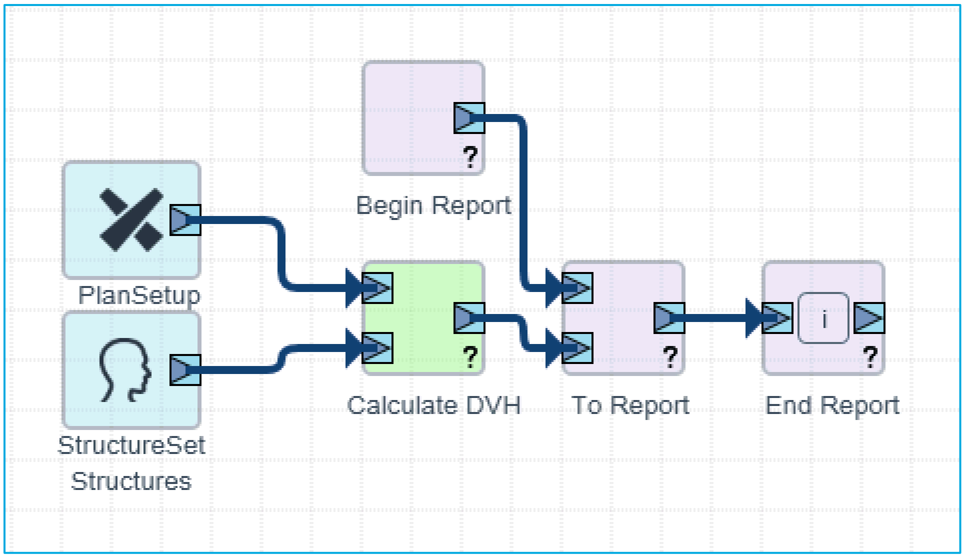


7. To define the file to export to, click the i on the **ToFile** action pack and enter a file name.
8. Define the metrics by clicking the i on the **Calculate DVH Metrics** action pack.
9. Use the syntax defined in the Mayo paper for the **DVH Objective** column.
10. You can define an Evaluator by defining Goal and Must criteria, if necessary. If an Evaluator is not defined, the metric will be calculated and reported, but not evaluated.
11. To add user-defined priority information to generated reports and tables, enter a value in the **Priority** column. The entered priority has no effect on processing or evaluation.
12. To map multiple structure IDs back to a single ID, use the Structure ID Dictionary. You can, for example, map “Femoral Head Rt” and “Femoral Head Right” to the alias id “fem_head_rt”.



Filter Structures Based on DICOM Type

The following example flow sends DVH data for all structures in the active **StructureSet** to a report.

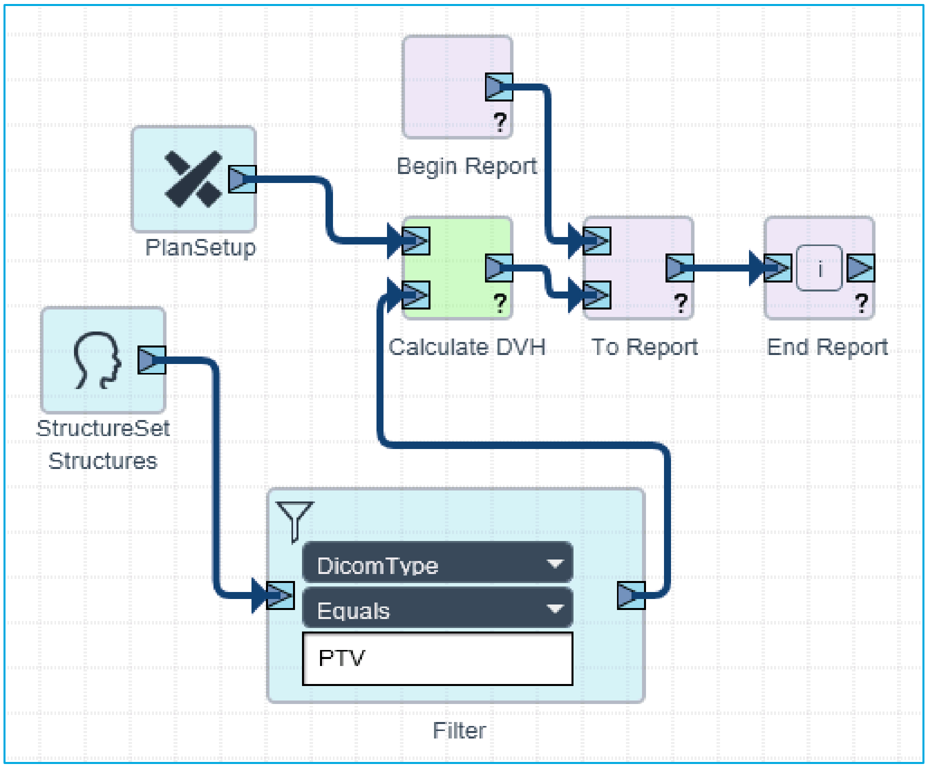


You can limit the flow so that DVH data for PTV type structures only is sent to the report by using the Filter flow control between the Structures context item and **CalculateDVH** action pack.

13. In Visual Scripting Workbench, drag all the action packs and context items illustrated in the image below to the canvas.
14. Add the required connections.

    Drag a Filter control between Structures and CalculateDVH.

15. Select DICOM Type and Equals and type PTV into the filter.

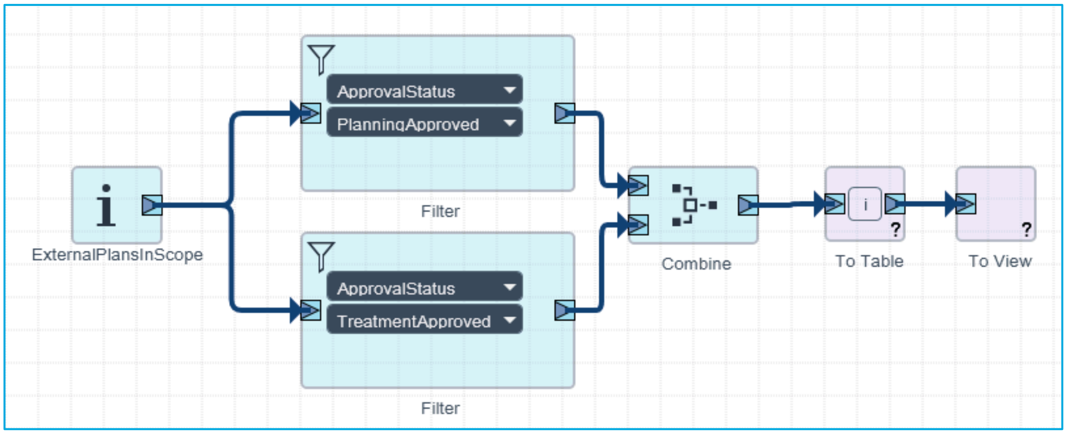


Filter and Combine Plans by Status

It is possible to use Filter and Combine flow controls together to create more complex visual script flows. You can do this, for example, to create a script that shows a list of all external beam plans with Planning Approved or Treatment Approved status. First you filter the plans to show only plans with planning-approved or treatment-approved status. Then you combine the filtered plans into a single table to view or report.

1. Drag the **ScriptContext** context item to the canvas and choose the sub-type **ExternalPlansInScope**.
2. Drag the Filter control to the canvas and for **ApprovalStatus**, select the value **PlanningApproved**. A list of planning-approved plans is created on the Output Port.
3. Add another filter for **ApprovalStatus** with the value **TreatmentApproved**. A list of treatment-approved plans is created on the Output Port.
4. Connect **ScriptContext** to the **Filter** controls.
5. Add the **Combine** control and connect it to the **Filter** controls. Two plans lists are combined.
6. Connect the **Combine** control to a new **ToTable** action pack. The combined plan list is sent from the **Combine** control to the **ToTable** action pack.
7. Configure the **ToTable** action pack by choosing properties of the plans you wish to display in a table (ID, for example) by clicking the i.

8. Add the **ToView** action pack. The selected properties from the plans are put into a table that is flowed to the **ToView** action pack where the plans list is shown to the user.



Developing Custom Action Packs for Visual Scripting

You can create your own custom action packs to be used in Visual Scripting. The action packs can be either read-only or write-enabled. Write-enabled action packs must be approved In Eclipse prior use in a clinical environment. Before using an action pack in a clinical system, you must follow the same professional software engineering and clinical development practices that you use for developing other scripts.

WARNING: The authors of custom scripts are responsible for verifying the accuracy and correctness of the scripts after developing a new script or after system upgrade for the existing scripts.

Create a Custom Action Pack with the Script Wizard

To create an action pack script with the Script Wizard, follow these guidelines:

1. From the **Start** menu, select **Varian > Eclipse Scripting API > Eclipse Script Wizard**.
2. Enter a name for the new action pack.
3. Select the Visual Scripting Action Pack option.
4. To select the location for storing the script, click **Browse**. By default, the action pack project is stored in the user-specific Documents folder.
5. Click **Create**.
6. The Script Wizard creates the following folder in the location that you selected if they do not already exist.
   - *Project* folder: Contains a script-specific sub-folder where the Microsoft Visual Studio project file and source code file are stored.

The Script Wizard launches Visual Studio.

7. Edit the source code as you wish.
8. Compile the plug-in by using Visual Studio (or MSBuild as described in Section *"Compile Example Scripts"*). The resulting action pack DLL is created in the *Plugins* folder.
9. Copy the custom action pack to the Visual Scripting Action Pack directory in *\\server\va_data$\ProgramData\Vision\VisualScripting\CustomActionPacks*.
10. If necessary, approve the action pack for use in Eclipse.
11. In the Visual Scripting Workbench, load the newly created custom action pack and create a flow that uses the new action pack.

To run the new script in Eclipse, choose **Menu > Save and Execute** in Eclipse.

Creating Scripts for Eclipse Automation

You can use the Eclipse Automation features in the Eclipse Scripting API for the tasks listed in this chapter.

Adding and Modifying Structures

You can add, remove, and modify structures by using the classes described below.

Adding and Removing Structures

Use the `StructureSet` class to add and remove structures:

- Add structures with the `AddStructure` method. You can use the `CanAddStructure` method to check if the script is able to add a new structure to the structure set.

  To specify the type of structure to add, give the string representation of the DICOM type as a parameter to the `AddStructure` method. Example values are EXTERNAL, ORGAN, and PTV. For a complete list of allowed values, refer to *Eclipse Scripting API Online Help*.
- Add structure code information with `AddStructure(StructureCodeInfo).` To construct `StructureCodeInfo`, you need to pass coding scheme and structure code, for example:
  ```
  StructureCodeInfo scInfo = new
  StructureCodeInfo("99VMS_STRUCTCODE", "Support");
  ```
- Remove structures with the `RemoveStructure` method.
- You can convert an isodose surface to a structure. See `Structure.ConvertDoseLevelToStructure`.

Modifying Structures

Use the `Structure` class to modify structures:

- Add contours to a structure with the `AddContourOnImagePlane` method.

  Input parameters: a list of points defining the contour, and the index of the image plane where the contours are to be added.
- Subtract contours with the `SubtractContourOnImagePlane` method.

  Input parameters: a list of points defining the contour, and the index of the image plane where the contours are to be subtracted.
- Clear all contours for a structure with the `ClearAllContoursOnImagePlane` method.
- Set a structure to the result of a Boolean operation with the And, Or, Not, and Xor methods. These methods are available also on the `SegmentVolume` class, which allows you to execute a combination of Boolean operations with an

intermediate variable of the `SegmentVolume` type before assigning the final result to a `Structure` object. For more information, see the `SegmentVolume` property getter and setter of the `Structure` class.

- Set a three-dimensional symmetric or asymmetric margin around a structure. First call the `Margin` or `AsymmetricMargin` method. Then set the resulting `SegmentVolume` object to the `Structure` object using the `SegmentVolume` property. For more information, see the get and set accessors of the `SegmentVolume` property of the `Structure` class.

Adding and Removing Artificial Phantom Images

Use the Patient class to add and remove artificial phantom images:

- Add a phantom image with the `AddEmptyPhantom` method.

  Input parameters: patient orientation, the size of the image set in X- and Y-directions in pixels and in millimeters, the number of planes, and the separation between the planes in millimeters.

  The image is created to a new `Study`. The return value of the `AddEmptyPhantom` is a new `StructureSet` where you can add structures and their contours. You can access the created Image using the Image property of the `StructureSet` class.

- Remove a phantom image and associated `StructureSet` with the `RemoveEmptyPhantom` method.

Input parameter: `StructureSet`. The `StructureSet` must not contain any `Structures`. Before removing the phantom, you can use the `CanRemoveEmptyPhantom` method to check if the script is able to remove the phantom `Image`.

Copying an Image from Another Patient

Use the `Patient` class to copy a 3D image from another patient. This must only be used when creating Verification Plans with a script.

- Specify the identifiers of the other patient, the study of the other patient, and the 3D image of the other patient. Validate the identifiers using the method `CanCopyImageFromOtherPatient`. Copy the image using the method `CopyImageFromOtherPatient`.

  Input parameters: the identifier of the other patient, the identifier of the study of the other patient, the identifier of the 3D image of the other patient.

Creating and Modifying Plans and Fields

You can create and modify external beam photon plans and fields by using the classes described below. For proton plans, you can only modify the scanning spot lists. Other modifications are not supported. For brachytherapy plans, no modifications are supported.

Adding and Removing Plans

Use the `Course` class to add and remove plans:

- Add a new external beam photon plan with the `AddExternalPlanSetup` method.

  Input parameter: `StructureSet`. A new primary reference point without a location is automatically created for the plan, or alternatively you can specify an existing reference point that will be used as primary. You can use the `CanAddPlanSetup` method to check if the script is able to add a new plan to the course.
- Add a new external beam photon plan as a verification plan with the `AddExternalPlanSetupAsVerificationPlan` method.

  Input parameters: structure set, the verified plan.
- Remove a plan with the `RemovePlanSetup` method.

  Input parameter: `PlanSetup`. You can use `CanRemovePlanSetup` to check if the script is able to remove the plan from the course.

Adding Fields

Use the `ExternalPlanSetup` class to add fields:

- Add a new static open field to the plan with the `AddStaticBeam` method.
- Add a new arc field to the plan with the `AddArcBeam` method.
- Add a field with static MLC shape with the `AddMLCBeam` or `AddMLCArcBeam` method.
- Add an arc field with dynamic MLC shape using the `AddConformalArcBeam` method.
- Add an IMRT field with the Multiple Static Segment delivery method using the `AddMultipleStaticSegmentBeam` method.
- Add an IMRT field with the Sliding Windows delivery method using the `AddSlidingWindowBeam` method.
- Add a VMAT field with the `AddVMATBeam` method.

Define the `TreatmentMachine` configuration for added fields:

- Create an `ExternalBeamMachineParameters` object and use it as input for the above mentioned beam adding methods of the `ExternalPlanSetup` class.
- Input parameters for the `ExternalBeamMachineParameters` object: The treatment machine identifier, energy mode identifier, field technique identifier, dose rate, and optionally, the primary fluence mode identifier.

For information about other input parameters for the beam adding methods of the `ExternalPlanSetup` class, refer to *Eclipse Scripting API Online Help*.

Modifying Fields

Use the `BeamParameters` class to modify a `Beam`:

- Call the `GetEditableParameters` method. Modify the returned `BeamParameters` object. Call `ApplyParameters`. If the call succeeds, the beam data is updated.
- Set the optimal fluence of an IMRT field with the `SetOptimalFluence` method.
- Modify `ControlPoints` using the `ControlPoints` property of the `BeamParameters` class. Each `ControlPoint` has one `ControlPointParameters` object. Call `ApplyParameters`. If the call succeeds, the beam data is updated.
- Methods to perform MLC fitting are available. See `Beam.FitMLCToStructure` and `Beam.FitMLCToOutline`.

Modifying Proton Scanning Spots

Use the `IonBeamParameters` class to modify the scanning spots of an `IonBeam`:

- Call the `IonBeam.GetEditableParameters` method.
- With the returned `IonBeamParameters` object, call property `IonControlPointPairs` to get a list of editable control point pairs.
- Iterate over the list of control point pairs and get the editable spot lists using method `IonControlPointPair.RawSpotList` or `IonControlPointPair.FinalSpotList`.
- Iterate over the scanning spot list and set the position or weight of each spot using setter properties `IonSpotParameters.Weight`, `IonSpotParameters.X` or `IonSpotParameters.Y`.
- When the scanning spots have been changed as desired, call `IonBeam.ApplyParameters(IonBeamParameters)`. If the call succeeds, the proton beam / control point / scanning spot data resident in Eclipse memory has been updated.

```
// unrealistic example shows how to edit the raw scanning spot lists
foreach (IonBeam ionBeam in context.IonPlanSetup.IonBeams)
{
  IonBeamParameters beamParams = ionBeam.GetEditableParameters();
  IonControlPointPairCollection cpList = beamParams.IonControlPointPairs;
  foreach (IonControlPointPair icpp in cpList)
  {
    IonSpotParametersCollection rawSpotList = icpp.RawSpotList;
    foreach (IonSpotParameters spot in rawSpotList)
    {
      spot.Weight = 1;  // set weight to desired
      spot.X = 1;       // set X position of scanning spot
      spot.Y = 1;       // set Y position of scanning spot
    }
  }
  // apply scan spot changes to Eclipse
  ionBeam.ApplyParameters(beamParams);
}
```

Figure 13 Sample Script Code to Edit Raw Scanning Spots

- Methods `IonControlPointPair.ResizeFinalSpotList` and `IonControlPointPair.ResizeRawSpotList` allow resizing the proton raw and final scanning spot lists.

Adding Prescriptions

- Set the prescription with the `SetPrescription` method of the `PlanSetup` object.

  Input parameters for the method: the number of fractions, the prescribed dose per fraction (using the dose unit defined for your system in the Varian Service Portal, and prescribed percentage.

Using Calculation Algorithms

You can use the classes described below for IMRT and VMAT optimization, leaf motion calculation, and dose calculation.

Setting Calculation Models

Use the `PlanSetup` class to set the calculation models:

- Set the calculation model with the `SetCalculationModel` method.

  Input parameters: calculation type and calculation model name.

  The current calculation model can be retrieved with the `GetCalculationModel` method.

  All available models can be read with the `GetModelsForCalculationType` method of the `ExternalPlanSetup` class.

- Set the calculation options using the `SetCalculationOption` method. The available options and their allowed values depend on the calculation model. Before setting the calculation options, set the calculation model.
- Clear the calculation model of a plan with the `ClearCalculationModel` method.

Viewing Calculation Logs

Use the following classes to read the messages from the algorithms:

- `Beam`

  Access the calculation logs after calculation with the `CalculationLogs` property. It returns a collection of `BeamCalculationLog` objects that can be filtered using the `Category` property. The category for the optimization log is, for example `Optimization`.

- `System.Diagnostics.Trace`

  Use the `Trace` class of the .NET framework to receive messages from the algorithms during the calculation. For more information about how to use `Trace Listeners`, consult the Microsoft Developer Network (MSDN) documentation.

Executing DVH Estimation

Use the `ExternalPlanSetup` class for executing DVH estimation.

- Run DVH estimation with the `CalculateDVHEstimates` method.

  Input parameters: the identifier of the DVH estimation model, the dose level for target structure(s) and the mapping between the structures of the estimation model and the structure set of the plan. Use the `Success` property of `OptimizerResult` to check if the algorithm executed without errors.

Optimizing IMRT and VMAT Plans

Use the following classes for optimizing IMRT and VMAT plans:

Setting up the Plan

- Before starting the optimization, create open fields for the plan with the `AddStaticBeam` or `AddArcBeam` method of the `ExternalPlanSetup` class.
- Use the `OptimizationSetup` property of the `PlanSetup` class to access the `OptimizationSetup` object.

Adding and Modifying Optimization Objectives

Use the `OptimizationSetup` object to add and modify optimization objectives:

- Add point objectives with the `AddPointObjective` method.

- Add mean dose or gEUD objectives with the `AddMeanDoseObjective` or `AddEUDObjective` methods.
- Add beam-specific parameters with the `AddBeamSpecificParameter` method.
- Add a Normal Tissue Objective with the `AddNormalTissueObjective` method.
- Remove optimization objectives with the `RemoveObjective` method.
- Remove optimization parameters with the `RemoveParameter` method.

Optimizing an IMRT Plan

Use the `ExternalPlanSetup` class to optimize an IMRT plan:

- Run the IMRT optimization algorithm with the `Optimize` method.

  Input parameter: the number of needed iterations. All existing optimal fluences are removed. As a result of Optimize, the `OptimizerResult` object is returned. Use the `Success` property of `OptimizerResult` to check if the algorithm executed without errors.
- You can continue IMRT optimization with the overloaded version of the `Optimize` method, for which you can give the `OptimizationOption` as a parameter. To continue optimization with existing optimal fluences, use the `OptimizationOption.ContinueOptimization` as a parameter.
- You can continue IMRT optimization with the existing plan dose as an intermediate dose. To do this, use the overloaded version of the `Optimize` method, and define the `OptimizationOption.ContinueOptimizationWithPlanDoseAsIntermediateDose` as a parameter.
- You can terminate IMRT optimization upon convergence. To do this, use the `Optimize` method that does not take any input parameters, or the overloaded version of the `Optimize` method that takes an `OptimizationOptionsIMRT` object as an input. Construct the `OptimizationOptionsIMRT` object using the `OptimizationConvergenceOption.TerminateIfConverged` option. After optimization you can read the actual number of iterations from `OptimizerResult.NumberOfIMRTOptimizerIterations`.
- You can use intermediate dose calculation during IMRT optimization. To do this, create an `OptimizationOptionsIMRT` object and specify the `numberOfStepsBeforeIntermediateDose`. Then use the overloaded version of `Optimize` method that takes an `OptimizationOptionsIMRT` object as an input.

Optimizing a VMAT Plan

Use the `ExternalPlanSetup` class to optimize a VMAT plan:

- Run the VMAT optimization algorithm with the `OptimizeVMAT` method. As a result of `Optimize`, the `OptimizerResult` object is returned. Use the `Success` property of `OptimizerResult` to check if the algorithm executed without errors.

- You can use intermediate dose calculation during VMAT optimization. To do this, create an `OptimizationOptionsVMAT` object by specifying either the option `OptimizationIntermediateDoseOption.UseIntermediateDose` or the number of optimization cycles. Then use the overloaded version of method `OptimizeVMAT` that takes an `OptimizationOptionsVMAT` object as an input parameter.

Accessing Dose Volume Histograms

Use the `OptimizerResult` class to access the results of the optimization:

- Use the `StructureDVHs` property to access the Dose Volume Histograms after optimization. The returned collection contains an `OptimizerDVH` object for each `Structure` that had optimization objectives defined. Use the `CurveData` property of the `OptimizerDVH` class to access the points of the Dose Volume Histogram.

Accessing Optimal Fluences after IMRT Optimization

Use the `Beam` class to access the optimal fluence information:

- Read the optimal fluence matrix after optimization with the `GetOptimalFluence` method.

Using Trade-Off Exploration

Use the `TradeoffExplorationContext` class and its methods and properties for exploring trade-offs in plans:

1. Start from an optimized and calculated plan by using the `TradeoffExplorationContext` class.
2. Select the trade-off objectives by using the `AddTradeoffObjective(Structure)` method. At least one objective is needed.
3. Query if all pre-conditions to generate the plan collection are met by calling the `CanCreatePlanCollection` property.
4. If `CanCreatePlanCollection` is true, generate the plan collection by calling the `CreatePlanCollection` method.
5. If the plan collection is generated successfully, the `HasPlanCollection` property is set to true. The class is now ready for exploring different trade-offs.
6. To evaluate the current trade-off, use the `GetObjectiveCost`, `CurrentDose`, and `GetStructureDvh` methods.
7. To explore different trade-offs:
   - Use the `SetObjectiveCost` method to reduce the cost of any objective.
   - Use the `SetObjectiveUpperRestrictor` method to prevent the cost of an objective from exceeding a specified limit.

8. Save the trade-off exploration results by calling the `ApplyTradeoffExplorationResult` method. The method also applies the trade-off exploration result to the plan setup for IMRT plans.
9. In case of a VMAT plan, call the `CreateDeliverableVmatPlan` method to apply the trade-off exploration result to the plan setup.
10. To resume the trade-off exploration from a saved plan collection, call the `LoadSavedPlanCollection` method.

Calculating Leaf Motions after IMRT Optimization

Use the `ExternalPlanSetup` class to execute the leaf motion calculation algorithm:

- Calculate leaf motions with the `CalculateLeafMotions` method. The method uses the default calculation options of the leaf motion calculation model set in the plan. The method returns a `CalculationResult` object. This object has a `Success` property, which you can use to check if the algorithm executed without errors.
- To run the Varian Leaf Motion Calculator algorithm, use the overloaded `CalculateLeafMotions` method.

  Input parameter: an `LMCVOptions` object. In `LMCVOptions`, you can specify the usage of fixed jaws. Check that the leaf motion calculation model of the plan is Varian Leaf Motion Calculator.
- To run the Varian Smart LMC algorithm, use the overloaded `CalculateLeafMotions` method.

  Input parameter: a `SmartLMCOptions` object. In `SmartLMCOptions`, you can specify the usage of fixed field borders and jaw tracking. Check that the leaf motion calculation model of the plan is Varian Smart LMC.
- To run the non-Varian MSS Leaf Motion Calculator algorithm, use the overloaded `CalculateLeafMotions` method.

  Input parameter: an `LMCMSSOptions` object. In `LMCMSSOptions`, you can specify the number of calculation iterations. Check that the leaf motion calculation model of the plan is MSS Leaf Motion Calculator.

Calculating Photon Plan Dose

Use the `ExternalPlanSetup` class for dose calculation:

- Calculate the volume dose using the `CalculateDose` method. The method returns a `CalculationResult` object. This object has a `Success` property that you can use to check if the algorithm executed without errors.
- Calculate the volume dose with preset MUs using the `CalculateDoseWithPresetValues` method.

  Input parameter: a list of `Beam` identifier and `MeterSetValue` pairs of the `Beam`.

Calculating Proton Plan Dose

Use the `IonPlanSetup` class for proton dose calculation:

- Calculate the volume dose using the `CalculateDose` method. The method returns a `CalculationResult` object. This object has a `Success` property that you can use to check if the algorithm executed without errors.

Creating an Evaluation Dose

Use the `ExternalPlanSetup` or `IonPlanSetup` class:

- Create a new `EvaluationDose` using the `CreateEvaluationDose` method. The method returns an `EvaluationDose` object. This object has a `SetVoxels` method that is called to set the content of the dose grid.
- Create a new `EvaluationDose` using the `CopyEvaluationDose` method. The method copies an existing `Dose` and returns it as an `EvaluationDose` object. Use the `SetVoxels` method to change the content of the dose grid.

Creating Halcyon Plans

Halcyon plans have additional requirements beyond standard photon plans before they can be approved. Halcyon plans need to have imaging setups and couch structures attached before they can be considered valid. To create a Halcyon plan with ESAPI automation, first create a plan with `Course.AddExternalPlanSetup`, then add an imaging setup with `ExternalPlanSetup.AddExternalPlanSetup`, and insert a couch with `ExternalPlanSetup. GenerateCouchStructures`. Check the validity of the plan with `ExternalPlanSetup. IsValidForPlanApproval`. See the code sample below:

```
public void Execute(ScriptContext context)
   {
       context.Patient.BeginModifications();
       Course c = context.Patient.AddCourse();
       c.Id = "halcyon";
       var ptv = context.StructureSet.Structures.First(s => s.DicomType ==
"PTV");
       var eps = c.AddExternalPlanSetup(context.StructureSet, ptv, ptv,
"refpt1");
       var machineparameters = new
ExternalBeamMachineParameters("Hal2_SX2_D5");
       var flat = eps.AddFixedSequenceBeam(machineparameters, 20, 45, new
VVector(0, 0, 0));
       var imagingparameters = new
ImagingBeamSetupParameters(ImagingSetup.MVCBCT_Low_Dose, 0, 0, 0, 0, 140,
140);
       eps.AddImagingSetup(machineparameters, imagingparameters, ptv);
       eps.GenerateCouchStructures("Halcyon Couch");

       eps.CalculateDose();

       IEnumerable<PlanValidationResultDetail> reasons;
```

```
        if (!eps.IsValidForPlanApproval(out reasons))
        {
            string message = "";
            foreach (PlanValidationResultDetail pvrd in reasons)
            {
                message += pvrd.MessageForUser + "\n";
            }
            MessageBox.Show("Halcyon plan is not valid for
approval.  Messages = \n" + message);
        }
        else
        {
            MessageBox.Show("Halcyon plan is valid for approval");
        }
    }
```

Approving Scripts for Clinical Use

This chapter describes how you can use script approval in Eclipse to support the script development at your clinic.

Script approval makes sure that all ESAPI scripts used clinically have been validated and approved by senior clinical personnel.

Approve a Script for Clinical Use

Write-enabled scripts must be approved before they can be used in a clinical system. A responsible senior level staff member can approve them in Eclipse (External Beam Planning and Plan Evaluation) and BrachyVision (Brachytherapy Planning and Brachytherapy 2D Entry). After the script has been properly coded and tested according to your script development process, do the following:

1. In Eclipse or BrachyVision, choose **Tools > Script Approvals** to open the **ESAPI Script Administration** window.
2. Copy the script to the clinical system and click **Register New Script** to register it in the system.
3. To approve for a limited evaluation, choose **Approve for Evaluation Testing**.
4. To approve for full clinical use for all clinical users, choose **Approve**.
5. Close the Script Approvals dialog box.
6. Select **File > Save All** to save changes.

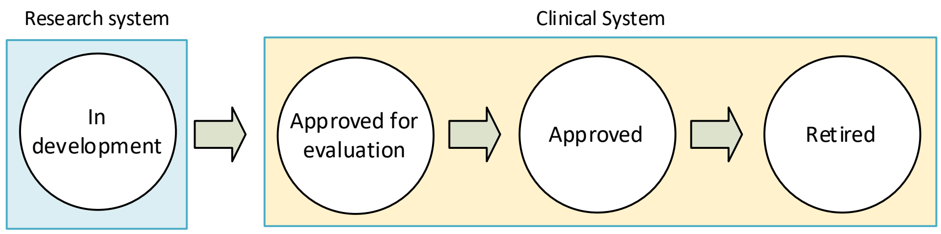
Script Development Process

Scripts are developed and tested in a non-clinical development system that allows running both read-only and write-enabled scripts without any approval process. When development is completed, scripts are moved to the clinical system, and to allow their use, scripts must be approved.

Script approval is mandatory for write-enabled scripts that use the Eclipse Automation features. For read-only scripts, approval is optional. Script approval is possible only for binary plug-in scripts and stand-alone executables. Any write-enabled custom action pack used in a visual script must be approved separately.

It is possible to first approve the script for evaluation use so that a group of users with specific evaluation rights can use the script for a limited time period in a clinical environment.

The following diagram illustrates the suggested steps in script development:



The system provides logging so that the QA responsible person can always find all plans and structure sets that have been modified by a write-enabled script.

Modifications made by a script to plans and structure sets are shown in the Plan Approval Wizard. The planner must acknowledge and approve the changes during the plan approval process.

For more information on configuration, see Chapter *”Approving Scripts for Clinical Use”,* Section *“Configuring a Non-Clinical Development System”*, and Section *“Configure a Clinical System to Require Approval for All Scripts”*.

Example Script Development Process

The following sections illustrate how script approval can be used to support the script development process in a radiotherapy clinic.

Initial Phase

- The lead Clinical Physicist in the Radiotherapy Department asks Programmer to develop a new script.
- Programmer starts creating the script following professional software engineering practices and the software development lifecycle in use at his institution.

Development in a Non-Clinical System

To develop the script, Programmer does the following work on the non-clinical development system:

- Works with Clinical Physicist to develop the user story and document the intended clinical use for the script.
- Writes testable user and system requirements for the script and reviews those with Clinical Physicist and other stakeholders.
- Analyzes potential risks and hazards with colleagues and documents a risk and hazards analysis for the script along with mitigators for identified potential hazards.
- Creates and documents a design for the script, if the script is large enough to warrant a design, and reviews that with stakeholders.
- Programs the script source code and reviews the source code with other programmers.

- Uses a software configuration management system to version and archive source code.
- Creates automated unit tests and verifies that near 100% code coverage is possible.
- Releases the script to Clinical Physicist and copies it to the release directory on the non-clinical development system.

Commissioning and Validation

Clinical Physicist does the following:

- Works with Programmer to create a commissioning and validation plan for the script.
- Validates and commissions the script for its intended clinical use and environment.
- After confirming that the script satisfies requirements and works as intended, creates a validation and commissioning report.
- Copies the script to the System Scripts directory on the clinical system.

Evaluation in a Clinical System

Clinical Physicist does the following:

- Signs in to the clinical system and opens the **ESAPI Script Administration** window.
- Chooses the newly developed script and sets the status to **Approved for Evaluation**.
- In Varian Service Portal, gives **Run Script in Evaluation State** user rights to two senior Treatment Planners, and notifies them that the script is available for evaluation.

Treatment Planners do the following:

- At Clinical Physicist's instruction, evaluate the script in the clinical system by shadowing actual clinical cases they are working on.
- Create evaluation courses with evaluation plans that are created with the script, and compare those evaluation plans against the actual clinical plans they have manually created.
- Complete the evaluation phase of the script, when they have successfully planned a number of different types of cases for different sites as defined in the validation and commission plan.

Clinical Physicist does the following:

- Reviews the results of the evaluation phase, and then finalizes the validation and commissioning report.

Approval for Clinical Use

Clinical Physicist does the following:

- Signs in to the clinical system and opens the **ESAPI Script Administration** window. Chooses the script and sets its status to **Approved**.
- The script is now available for routine clinical use by all treatment planners in the department.

Differences between Clinical and Non-Clinical Environments

Non-clinical environments (also called research environments) are different from clinical environments in that:

- Users cannot treatment-approve plans, which prevents treating from a research environment database.
- The title bars in Eclipse workspaces note that the system is running in a research environment.
- All Eclipse printouts generated from this environment note that they were created in a research environment.
- The Beam Configuration application warns that changes should only be made if the DCF environment is dedicated for research.
- All plans created or modified through the Eclipse Scripting API have their intent set to “Research”.
- Users can execute unapproved scripts for debugging and testing purposes.

Configuring a Non-Clinical Development System

When Eclipse is configured as a non-clinical development system (also called research system), additional API features are available for non-clinical research and development use.

A development environment has an Eclipse Scripting API research license installed, and the Varian System database has been configured for research use. The configuration is typically done by Varian service personnel at time of installation.

Configure Eclipse for Non-Clinical Use

To configure an Eclipse system for non-clinical development use (assuming you have the proper user rights and the Eclipse Scripting API for Research Users license):

1. Open **RT Administration** using a system administrator account.
2. Click **System and Facilities**.
3. Click **System Properties**.
4. Select the **Database in Research Mode** check box.
5. Select **File > Save All** to save changes.

Tip: You can use the following script to test whether write-enabled scripts can be run on the system.

```
using System;
using VMS.TPS.Common.Model.API;
using VMS.TPS.Common.Model.Types;
using System.Windows;
[assembly: ESAPIScript(IsWriteable = true)]
namespace VMS.TPS
{
  public class Script
  {
    public void Execute(ScriptContext context)
    {
      if (context.Patient == null)
      {
        MessageBox.Show("Please load a patient before running this script.");
        return;
      }
      try
      {
        // throws an exception if writable scripting not enabled.
        context.Patient.BeginModifications();
        MessageBox.Show("SUCCESS!  Writable scripting is enabled.");
      }
      catch (Exception e)
      { string Message = "Test FAILED!  Writable scripting is not enabled.\n";
        Message += "Message:\n" + e.Message;
        MessageBox.Show(Message);
      }
    }
  }
}
```

Configure a Clinical System to Require Approval for All Scripts

The Eclipse system can be configured to enforce script approvals for all script types, read-only and write-enabled. By system design, approval is mandatory for write-enabled scripts. For read-only scripts you can decide whether to make the approval mandatory or not.

To configure the system to require approvals for all script types, both read-only and write-enabled:

1. Open **RT Administration** using a system administrator account.
2. Click **System and Facilities**.
3. Click **System Properties**.
4. Select the **Approval** required for read-only scripts check box.
5. Select **File > Save All** to save changes.

Release a New Version of a Script that Is in Clinical Use

Before releasing a new version of a script that is in clinical use, retire the current version of the script:

1. In the **Script Approvals** dialog box, select the current version of the script and click **Retire**.
2. Enter your user name and password, and click **Retire**.
3. Approve the new version of the script following instructions in the prior section.
4. Close the **Script Approvals** dialog box.
5. Select **File > Save All** to save changes.

Find All Plans and Structure Sets Changed by a Script

You may find, for example, that a write-enabled script you created and approved produces erroneous results due to a coding defect, and the script has been approved and used clinically for some time.

To evaluate the impact of the defect, you can create a stand-alone executable script that uses the API methods `PlanSetup.ApplicationScriptLogs` and `StructureSet.ApplicationScriptLogs` to find all plans and structure sets that the script has changed in the system. The following code sample shows an implementation of a stand-alone executable `Execute` method that could do this. Note that a stand-alone executable script like this opens every patient in the database and should only be run after-hours when no other users are working.

```
static void Execute(Application app)
{
  const string MyScript = "MyScript.esapi";
  // look for objects created since April 1st 2016 since script was
  // approved then.
  DateTime searchSince = new DateTime(2016, 4, 1);
  double searchDays = (DateTime.Now - searchSince).TotalDays;

  foreach (PatientSummary summary in app.PatientSummaries)
  {
    Patient p = app.OpenPatient(summary);
    // find all plans touched by the script
    foreach (Course course in p.Courses)
    {
      foreach (PlanSetup plan in
          course.PlanSetups.Where
          (ps => (DateTime.Now-ps.HistoryDateTime).TotalDays <= searchDays))
      {
        var scriptLogs = plan.ApplicationScriptLogs;
        foreach (ApplicationScriptLog log in
            scriptLogs.Where(s => s.ScriptFullName == MyScript))
        {
          Console.WriteLine("Plan \"{0}/{1}/{2}\" touched by " +
                            "script {3} on {4}.",
              p.Id, course.Id, plan.Id,
          log.ScriptFullName, log.HistoryDateTime);
        }
      }
    }
    // find all structure sets touched by the script
    foreach (StructureSet structSet in
        p.StructureSets.Where
        (ss => (DateTime.Now - ss.HistoryDateTime).TotalDays <= searchDays))
    {
      var scriptLogs = structSet.ApplicationScriptLogs;
      foreach (ApplicationScriptLog log in
          scriptLogs.Where(s => s.ScriptFullName == MyScript))
      {
        Console.WriteLine("Structure set \"{0}/{1}\" touched by " +
                          "script {2} on {3}.",
            p.Id, structSet.Id,
            log.ScriptFullName, log.HistoryDateTime);
      }
    }
    app.ClosePatient();
  }
}
```

Information about Used Scripts in Plan Approval

The **Plan Approval Wizard** contains the following information about the script that has been used to modify the plan or its structure set:

- Name of the script.
- Version of the script.
- Time when the plan or structure set was first saved after the modifications.

If you copy a plan or structure set that has been modified by a script, the same modification information is also transferred to the copied plan or structure set, and shown in the **Plan Approval Wizard**.

However, in this case, the time when you first saved the copied plan or structure set is shown instead of the time when the original plan or structure set was saved.

Using Scripts in Eclipse

You can launch plug-in scripts from the Tools menu in Eclipse (External Beam Planning, Plan Evaluation) and BrachyVision (Brachytherapy Planning, Brachytherapy 2D Entry). You can also store scripts as favorites in the same menu.

Stand-alone executables can be launched as any Windows application.

Launch a Plug-in Script

To launch a plug-in script:

1. Choose **Tools > Scripts**. The Scripts dialog box opens.
2. To locate the script that you want to run, select one of the following options:
   - **System Scripts**: The scripts that are available for all users are shown on the list.
   - **Directory: [path_to_your_own_scripts]**. Click **Change Directory** and select a folder. All files with the .cs or .esapi.dll file name extension become available on the list.
3. In the Scripts dialog box, select the script file on the list.
4. Click **Run**.
5. If the execution of the script takes a very long time, you can click the **Abort** button. The execution of the script is aborted the next time the script accesses a property or method of the Eclipse Scripting API. Note that this procedure is meant only for recovering from programming errors and should not be considered a normal practice.

Launch a Stand-Alone Executable Application

You can launch a stand-alone executable like any Windows application on the workstation where Eclipse is installed. You can also debug the stand-alone executable using normal Windows debugging tools.

Add and Remove Favorite Scripts

You can add favorite scripts to the Tools menu and define keyboard shortcuts for them.

Add a Favorite Script to the Tools Menu

1. Choose **Tools > Scripts**. The Scripts dialog box opens.
2. Select the script that you want to add to the menu.
3. Click **Add…**
4. A dialog box is opened. You can define a keyboard shortcut for the favorite script.
5. Click **OK**.

Remove a Favorite Script from the Tools Menu

1. In Eclipse, select **Tools > Scripts**. The Scripts dialog box opens.
2. Select a favorite script.
3. Click **Remove**.

Launch a Visual Script

Visual scripts can be run from within the Visual Scripting Workbench. You can also store scripts as favorites there. See Chapter "*Creating Visual Scripts"*, Section "*Create and Test a Visual Script"*.

In addition, visual scripts can be run in Eclipse like any other plug-in script (see Chapter *"Using Scripts in Eclipse",* Section *"Launch a Plug-in Script"*). You can also store visual scripts as favorites in the Tools menu.

The system requires that all write-enabled action packs used in a visual script must be approved for use before script execution.